CDC 00594

IMPORTANT NEWS ON PAGE 3

# The Transactor

The Tech/News Journal For Commodore Computers

95% Advertising Free! July 1987: Volume 8, Issue 01. $3.50

## Math

- Function Manipulation: Roots & Integrals
- Array Math Operations on the C64
- A Faster Square Root for the C64
- Complex Number Arithmetic
- Floating Point Accumulator Facts
- High-Speed Multiplies and Divides
- A Real Shuffle Subroutine
- PoptoFront & TrapSnapper for Amiga
- Speeding Up IFF File Access
- C-128 80-Column Interrupt Routines
- C-128 Bullet-Proof Windows
- Cross Reference Programmers Utility
- The Ultimate Compare Utility
- SymLister for the SYMASS Assembler

J. Mostacci LC

FREE! T-Shirt Offer see page 76

0 72246 00594 0 7

# THE TIME SAVER

**Type in a lot of Transactor programs?**
**Does the above time and appearance of the sky look familiar?**
**With The Transactor Disk, any program is just a LOAD away!**

**Only $8.95 Per Issue**
**6 Disk Subscription (one year)**
**Just $45.00**
**(see order form at center fold)**

**Also check out the TransBASIC Disk**
**Complete with 24 page manual, just $9.95!**

# Volume 8
# Issue 01

Circulation at Large
72,000

## Math


**Start Address** Editorial . . . . . 3

**Bits and Pieces** . . . 6
C-128 Easy Program Loading
C-128 80-Column Interrupt Routines
C-128 CAPS-lock q Fix
C-128 Bullet-Proof Windows
C-128 80-Column Display on a Television
1541 Half-Track Fix
The Drivelight Zone
Obscure C-64 VAL bug
G-Link ROM Compare
Input Trickery
C-64 Efficient Keyboard-Checking in Assembler
C-64 Scrolling Banner Routine
C-64 Undocumented Editing Mode
Easy Input Speedup from BASIC
Systematic Loop-Variable Naming
Selective Scratching
Amiga Automatic Up-Date
Pattern Matching With The Dir Command
Multiple-menu Selections
Speeding Up IFF File Access

**TransBloopurz** . . 20
Cap Meter Bits
N-Body Simulator on Transactor Disk 17
Some Notes on Transactor Disk 17
A Two-Button Mouse

**Letters** . . . . . 13
Wanted – Plus/4 Technical Info
Wanted – an IMG stripper for tape
A lead on cables
Another IEEE for the 128
Ultimate frustration fix
Biblical Greek drills
Hardware book blurb
Grounding to a halt
Holes in Inner Space
Structuring Basic programs
FOG clarifies
Simplifying the raid
Double Density HR S&P
"Fast File" relocation
A dissenting vote for C Power
C Converter for Super-C v1
Super C and the RAM disk
TransBASIC graphing on one screen
Bundling TransBASIC dialects
Amiga music software shortage
More Superkit sorrows

**News BRK** . . . . . 76
New Address and Phone Number
Subscription Intersection Set
Disk Subscription Notes
Free Transactor T's with Mag + Disk Sub.
Subscriber Mail Orders
Customs/Duty on Hardware Products
Transactor Mail Order
National Computer Conference 1987
4040 Drive Internals
GEOS Programming Guide
New Commodore Business Magazine
Genealogy Software from ByteWare
The MailRoom v2.1 for the Commodore 64
Spence XP BBS for C64 – $10
SpeedScript Updated for the C-128
Disk-2-Disk from CCS
New Interface for C-64/C-128 and IBM PC
Command Center for Commodore computers

Telecolumn . . . . . 21
A Real Shuffle Subroutine Shuffles cards the way humans do . . . . . 24
Function Manipulation: Roots and Integrals . . . . . 25
Array Math Operations on the C64 Simplify array calculations . . . 28
A Faster Square Root for the C64 More accurate, too . . . . . 34
Complex Number Arithmetic Add complex number data-types to BASIC . 36
FAC1 Facts Machine Language floating-point made easy . . . . . 40
High-speed Multiplies and Divides ML math routines explained . . 42
Secondary Address Bits Little-known tips from Jim Butterfield . . . . . 44
Interfacing and Controlling the Armatron Robot . . . 46
SymLister A list formatter for the Symass assembler . . . . . 50
XREF Variable and line-number cross referencing in a single command . . . . . 52
Beyond Compare A multi-format compare/display utility . . . . . 56

### Amiga Section:

PoptoFront An alternative to the electronic shell-game . . . . . 65
TrapSnapper Stop that Guru before he stops you! . . . . . 68
Amiga Dispatches The A2000 and 500, news, and some programming insights . 71
Compu-toons . . . . . 75

The Transactor
The Tech/News Journal For Commodore Computers

**Athos Editor**
Karl J. H. Hildon

**Porthos Editor**
Richard Evers

**Aramis Editor**
Chris Zamara

**D'Artagnan Editor**
Nick Sullivan

**Art Director**
John Mostacci

**Administration & Subscriptions**
Anne Richard
Kathryn Holloway

**Contributing Writers**

Ian Adam
David Archibald
Jim Barbarello
Anthony Bertram
Tim Bolbach
Ranjan Bose
Donald Branson
Anthony Bryant
Jim Butterfield
Betty Clay
Joseph Caffrey
Gary Cobb
Tom K. Collopy
Robert V. Davis
Elizabeth Deal
Frank E. DiGioia
Chris Dunn
Michael J. Erskine
Jack Farrah
William Fossett
Jim Frost
Miklos Garamszeghy
Eric Guiguere
Thomas Gurley
R. James de Graff
Tim Grantham
Adam Herst
Thomas Henry
John Holttum
David Hook
John Houghton
Robert Huehn
David Jankowski
Brian Junker
Clifton Karnes
Lorne Klassen
Jesse Knight
Gregory Knox
David Lathrop
James A. Lisowski
Richard Lucas
Scott Maclean
David Martin
Steve McCrystal
Stacy McInnis
Chris Miller
Terry Montgomery
Ralph Morrill
Rick Morris
Michael Mossman
Bryce Nesbitt
Gerald Neufeld
Noel Nyman
Kevin O'Connor
Richard Perrit
Terry Pridham
Raymond Quirling
Richard Richmond
Gary Royal
John W. Ross
E.J. Schmahl
David Shiloh
P. A. Slaymaker
John Spencer
Darren J. Spruyt
Aubrey Stanley
David Stidolph
Richard Stringer
Anton Treuenfels
Audrys Vilkas
Jack Weaver
Evan Williams

**Production**
Attic Typesetting Ltd.

**Printing**
Printed in Canada by
MacLean Hunter Printing

## Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') is a straight line as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print ' ' flush right ' ' – would be shown as – print ' '[10 spaces]flush right ' '

### Cursor Characters For PET / CBM / VIC / 64

**Down** – [q]
**Up** – [Q]
**Right** – [ ]
**Left** – [Lft]
**RVS** – [r]
**RVS Off** – [R]
**Insert** – [T]
**Delete** – [t]
**Clear Scrn** – [S]
**Home** – [s]
**STOP** – [c]

### Colour Characters For VIC / 64

**Black** – [P]
**White** – [e]
**Red** – [2]
**Cyan** – [Cyn]
**Purple** – [Pur]
**Green** – [5]
**Blue** – [←]
**Yellow** – [Yel]
**Orange** – [A]
**Brown** – [U]
**Lt. Red** – [V]
**Grey 1** – [W]
**Grey 2** – [X]
**Lt. Green** – [Y]
**Lt. Blue** – [Z]
**Grey 3** – [Gr3]

### Function Keys For VIC / 64

**F1** – [E]
**F2** – [I]
**F3** – [F]
**F4** – [J]
**F5** – [G]
**F6** – [K]
**F7** – [H]
**F8** – [L]

**Please Note: The Transactor's NEW phone number is: (416) 737-2786**

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number **6342.** USPS **725-050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209 **ISSN# 0827-2530.**

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64) are registered trademarks of Commodore Inc.

Subscriptions:
Canada $15 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

**Send all subscriptions to:** The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 878 8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

U.S.A. Distributor:
Capital Distributing
Charlton Building
Derby, CT
06418
(203) 735 3381
(or your local wholesaler)

**Quantity Orders:**

Master Media
261 Wyecroft Road
Oakville, Ontario
L6J 5B4
(416) 842 1555
(or your local wholesaler)

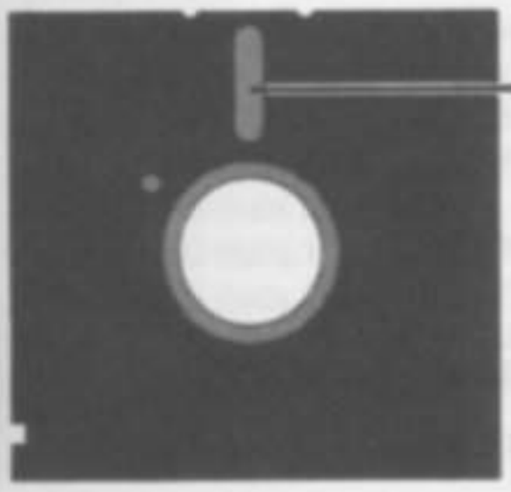
Norland Communications
251 Nipissing Road, Unit 3
Milton, Ontario
L9T 4Z5
416 876 4774

**SOLD OUT:** The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 03, 04, 05, 06, and Vol 5 Issues 02, 03, 04 are available on microfiche only
**Still Available:** Vol. 4: 01, 02. Vol. 5: 01, 04, 05, 06. Vol. 6: 01, 02, 03, 04, 05, 06. Vol. 7: 01, 02, 03, 04, 05, 06. Vol. 8: 01

**Back Issues:** $4.50 each. Order all back issues from Milton HQ.

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is $40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos or illustrations will be included with articles depending on quality. Authors submitting diskettes will receive the Transactor Disk for the issue containing their contribution.

All material accepted becomes the property of The Transactor. All material is copyright by Transactor Publications Inc. Reproduction in any form without permission is in violation of applicable laws. Please re-confirm any permissions granted prior to this notice. Solicited material is accepted on an all rights basis only. Write to the Milton address for a writers package.
The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.



# Start Address

## Our Last Newsstand Issue!

It's been just over eight years that I've been making Transactors, and except for a couple of newsletters from Commodore when their HQ was Palo Alto, and one or two out of Commodore in Japan, The T. was the first Commodore periodical. At the time Gene Beales was doing PET User Notes and Len Lindsay was making The PET Gazette (which would become COMPUTE! Magazine). But aside from the three, information was scarce and the big guys printed Commodore related articles on occasion only.

Then there was the explosion. COMPUTE and Gazette had strong footholds, and Commodore started publishing two slick efforts of their own. Other independent publishers followed. But The Transactor would stay a humble newsletter for another year. On June 1, 1982, almost five years ago, Transactor finally broke out of its shell to attempt the impossible: competition at the newsstand.

I learned how to typeset in a month, and shipped Volume 4, Issue 01 to a handfull of magazine racks that October. Surprisingly, we did well – "surprising" because The T. was still just a newsletter, except it had a colour cover and the articles were typeset instead of printed on the NEC. I have to laugh when I leaf through one of those old issues, kinda like looking at your first program in BASIC. I wasn't satisfied when an article didn't end at the bottom of a page, but at the time I didn't have the skill to do better. We store each issue on 256K, 8" floppies. That first issue needed one – this issue we went through five!

But the information was solid and the Toronto area had a high concentration of technocrats looking to assimilate *and* disseminate. In fact, the content may have been too solid.

The following June we approached Master Media, a national newsstand distributor. They were skeptical about shipping another computer title to potentially saturated shelves, but decided to give us a go anyway. Their U.S. counterpart, Capitol Distributing, didn't take us on for the same reason. But after the Canadian market showed sales steadily improving, they reconsidered. In June '84, 16,000 Transactors would find their way to stands across the U.S., and some even to Europe, S.America, and Australia.

Orders came piling in. Both Master Media and Capitol increased their orders every issue and would sell 50% of the copies or more (30% sales at the newsstand is considered excellent by publishers). The newsstand print order peaked at 53,000.

Naturally, the intent was to generate subscriptions – the mainstay for any magazine. They did increase, but fell short of expectations. The "solid" information we had to offer had a much smaller market than the glossy full-colour material from our competitors. However, we were confident that subscriptions would be generated by those advancing beyond games and other pre-packaged hobbies. That happened too. But again, the proportion of those who lose interest completely is much greater than those who transcend to a fascination with the science of computers.

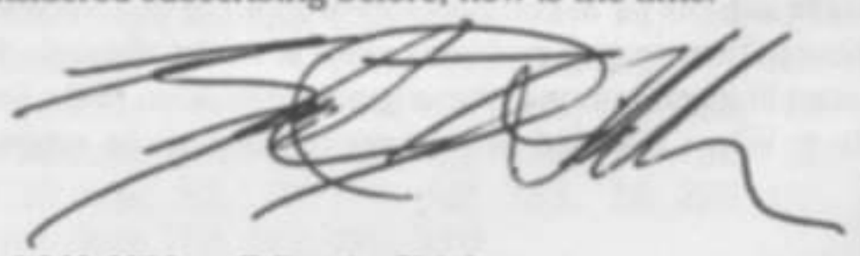
Many things have happened over the years (which seem like months), and within a month another milestone will be planted. Myself, Richard, Chris and Nick have placed an offer with our parent company to purchase Transactor Publishing Inc. The offer has all but been signed. We take possession after this issue is shipped, and you'll see that new addresses are on the back of our subscription card. Our new HQ is a 1500 square foot unit in a professional plaza just North of Toronto, and the builders should be starting work on our two-storey office layout any day.

Unfortunately, the newsstand distribution business is just too tough. Standard payment terms are 120 days from the "off-sale" date. Since we publish once every two months, it takes 6 months from the day an issue is pressed to the day we receive payment for the last copy sold. Our printer won't wait that long, and between the four of us, we can't float that kind of financing. So unless we can find another method for getting Transactors to the magazine racks, this will be our last newsstand appearance.

However, the newsstand sales do not generate a lot of revenue. In fact, they're not supposed to. That may sound absurd, but like I said, newsstand sales are supposed to generate subscriptions. A publishers' main goal at the newsstand is to break even. Advertising rates are driven by total sales, and *that's* where the profit lies. Of course advertising revenue at The T. accounts for only a small deposit at the bank. However, between subscriptions, disks, and our ancillary products, income from CompuServe, and TPUG's regular purchase, we've managed to do rather well. AND, we expect to continue doing well!

There will be other changes, mostly cost cutting. The paper we print on is the second most expensive available. We won't be regressing to newsprint or anything, but the familiar coated stock used in most publications actually costs less, and is more appealing to advertisers. Yep, advertisers. And if anyone you know is interested in ad space, we're all ears! Even if interest is high, though, I guarantee at least a meg of type per issue as usual.

But according to my calculations, there's at least 25,000 of you listening to me blabber right now that are not currently subscribing. And we need your help. Besides, you won't be able to get Transactors any other way, they'll come right to your door, and you'll be getting them for less than they cost you now. Our next issue will feature *operating systems and it's looking like a good one. Porthos, Aramis,* D'Artagnan, and myself want to keep smashing out Transactors, hopefully as much as you want to keep getting them! So if you've ever considered subscribing before, now *is* the time!



Karl J.H. Hildon, Editor in Chief

# Using "VERIFIZER"

## *The Transactor's Foolproof Program Entry Method*

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two–letter code for each program line which you can check against the corresponding code in the program listing.

There are five versions of VERIFIZER here; one for PET/CBMs, VIC or C64, Plus 4, C128, and B128. Enter the applicable program and RUN it. If you get a data or checksum error, re–check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 634 to enable the PET/CBM version (off: SYS 637)
SYS 828 to enable the C64/VIC version (off: SYS 831)
SYS 4096 to enable the Plus 4 version (off: SYS 4099)
SYS 3072,1 to enable the C128 version (off: SYS 3072,0)
BANK 15: SYS 1024 for B128 (off: BANK 15: SYS 1027)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

**Note:** If a report code is missing (or "––") it means we've edited that line at the last minute which changes the report code. However, this will only happen occasionally and usually only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re–check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors like POKE 52381,0 instead of POKE 53281,0. However, VERIFIZER uses a "weighted checksum technique" that can be fooled if you try hard enough; transposing two sets of 4 characters will produce the same report code but this should never happen short of deliberately (verifizer could have been designed to be more complex, but the report codes would need to be longer, and using it would be more trouble than checking code manually). VERIFIZER ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

**Technical info:** VIC/C64 VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm–start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

**PET/CBM VERIFIZER (BASIC 2.0 or 4.0)**

```
CI  10 rem* data loader for "verifizer 4.0" *
CF  15 rem pet version
LI  20 cs=0
HC  30 for i=634 to 754:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
OG  60 if cs<>15580 then print"***** data error *****":end
JO  70 rem sys 634
AF  80 end
IN  100 :
ON  1000 data  76,138,  2,120,173,163,  2,133,144
IB  1010 data 173,164,  2,133,145, 88, 96,120,165
CK  1020 data 145,201,  2,240, 16,141,164,  2,165
EB  1030 data 144,141,163,  2,169,165,133,144,169
HE  1040 data   2,133,145, 88, 96, 85,228,165,217
OI  1050 data 201, 13,208, 62,165,167,208, 58,173
JB  1060 data 254,  1,133,251,162,  0,134,253,189
PA  1070 data   0,  2,168,201, 32,240, 15,230,253
HE  1080 data 165,253, 41,  3,133,254, 32,236,  2
EL  1090 data 198,254, 16,249,232,152,208,229,165
LA  1100 data 251, 41, 15, 24,105,193,141,  0,128
KI  1110 data 165,251, 74, 74, 74, 74, 24,105,193
EB  1120 data 141,  1,128,108,163,  2,152, 24,101
DM  1130 data 251,133,251, 96
```

**VIC/C64 VERIFIZER**

```
KE  10 rem* data loader for "verifizer" *
JF  15 rem vic/64 version
LI  20 cs=0
BE  30 for i=828 to 958:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FH  60 if cs<>14755 then print"***** data error *****":end
KP  70 rem sys 828
AF  80 end
IN  100 :
EC  1000 data  76, 74,  3,165,251,141,  2,  3,165
EP  1010 data 252,141,  3,  3, 96,173,  3,  3,201
OC  1020 data   3,240, 17,133,252,173,  2,  3,133
MN  1030 data 251,169, 99,141,  2,  3,169,  3,141
MG  1040 data   3,  3, 96,173,254,  1,133, 89,162
DM  1050 data   0,160,  0,189,  0,  2,240, 22,201
CA  1060 data  32,240, 15,133, 91,200,152, 41,  3
NG  1070 data 133, 90, 32,183,  3,198, 90, 16,249
OK  1080 data 232,208,229, 56, 32,240,255,169, 19
AN  1090 data  32,210,255,169, 18, 32,210,255,165
GH  1100 data  89, 41, 15, 24,105, 97, 32,210,255
JC  1110 data 165, 89, 74, 74, 74, 74, 24,105, 97
EP  1120 data  32,210,255,169,146, 32,210,255, 24
MH  1130 data  32,240,255,108,251,  0,165, 91, 24
BH  1140 data 101, 89,133, 89, 96
```

**VIC/64 Double Verifizer** **Steven Walley, Sunnymead, CA**

When using 'VERIFIZER' with some TVs, the upper left corner of the screen is cut off, hiding the verifizer–displayed codes. DOUBLE VERIFIZER solves that problem by showing the two–letter verifizer code on both the first and second row of the TV screen. Just run the below program once the regular Verifizer is activated.

```
KM 100 for ad=679 to 720:read da:poke ad,da:next ad
BC 110 sys 679: print: print
DI 120 print "double verifizer activated":new
GD 130 data 120, 169, 180, 141,  20,   3
IN 140 data 169,   2, 141,  21,   3,  88
EN 150 data 96,  162,   0, 189,   0, 216
KG 160 data 157,  40, 216, 232, 224,   2
KO 170 data 208, 245, 162,   0, 189,   0
FM 180 data   4, 157,  40,   4, 232, 224
LP 190 data   2, 208, 245,  76,  49, 234
```

**VERIFIZER For Tape Users** **Tom Potts, Rowley, MA**

The following modifications to the Verifizer loader will allow VIC and 64 owners with Datasettes to use the Verifizer directly (without the loader). After running the new loader, you'll have a special copy of the Verifizer program which can be loaded from tape without disrupting the program in memory. Make the following additions and changes to the VIC/64 VERIFIZER loader:

```
NB  30   for i=850 to 980: read a: poke i,a
AL  60   if cs<>14821 then print "*****data error*****": end
IB  70   rem sys850 on, sys853 off
--  80   delete line
--  100  delete line
OC 1000 data  76,  96,   3, 165, 251, 141,   2,   3, 165
MO 1030 data 251, 169, 121, 141,   2,   3, 169,   3, 141
EG 1070 data 133,  90,  32, 205,   3, 198,  90,  16, 249
BD 2000 a$="verifizer.sys850[space]"
KH 2010 for i=850 to 980
GL 2020 a$=a$+chr$(peek(i)): next
DC 2030 open 1,1,1,a$: close 1
IP 2040 end
```

Now RUN, pressing PLAY and RECORD when prompted to do so (use a rewound tape for easy future access). To use the special Verifizer that has just been created, first load the program you wish to verify or review into your computer from either tape or disk. Next insert the tape created above and be sure that it is rewound. Then enter in direct mode: OPEN1:CLOSE1. Press PLAY when prompted by the computer, and wait while the special Verifizer loads into the tape buffer. Once loaded, the screen will show FOUND VERIFIZER.SYS850. To activate, enter SYS 850 (not the 828 as in the original program). To de-activate, use SYS 853.

If you are going to use tape to SAVE a program, you must de-activate (SYS 853) since VERIFIZER moves some of the internal pointers used during a SAVE operation. Attempting a SAVE without turning off VERIFIZER first will usually result in a crash. If you wish to use VERIFIZER again after using the tape, you'll have to reload it with the OPEN1:CLOSE1 commands.

**Plus 4 VERIFIZER**

```
NI 1000 rem * data loader for "verifizer +4"
PM 1010 rem * commodore plus/4 version
EE 1020 graphic 1: scnclr: graphic 0: rem make room for code
NH 1030 cs=0
JI 1040 for j=4096 to 4216: read x: poke j,x: ch=ch+x: next
AP 1050 if ch<>13146 then print "checksum error": stop
NP 1060 print "sys 4096: rem to enable"
JC 1070 print "sys 4099: rem to disable"
ID 1080 end
PL 1090 data  76,  14,  16, 165, 211, 141,   2,   3
CA 1100 data 165, 212, 141,   3,   3,  96, 173,   3
OD 1110 data   3, 201,  16, 240,  17, 133, 212, 173
LP 1120 data   2,   3, 133, 211, 169,  39, 141,   2
EK 1130 data   3, 169,  16, 141,   3,   3,  96, 165
DI 1140 data  20, 133, 208, 162,   0, 160,   0, 189
LK 1150 data   0,   2, 201,  48, 144,   7, 201,  58
GJ 1160 data 176,   3, 232, 208, 242, 189,   0,   2
DN 1170 data 240,  22, 201,  32, 240,  15, 133, 210
GJ 1180 data 200, 152,  41,   3, 133, 209,  32, 113
CB 1190 data  16, 198, 209,  16, 249, 232, 208, 229
CB 1200 data 165, 208,  41,  15,  24, 105, 193, 141
PE 1210 data   0,  12, 165, 208,  74,  74,  74,  74
DO 1220 data  24, 105, 193, 141,   1,  12, 108, 211
BA 1230 data   0, 165, 210,  24, 101, 208, 133, 208
BG 1240 data  96
```

**C128 VERIFIZER (40 column mode)**

```
PK 1000 rem * data loader for "verifizer c128"
AK 1010 rem * commodore c128 version
JK 1020 rem * use in 40 column mode only!
NH 1030 cs=0
OG 1040 for j=3072 to 3214: read x: poke j,x: ch=ch+x: next
JP 1050 if ch<>17860 then print "checksum error": stop
MP 1060 print "sys 3072,1: rem to enable"
AG 1070 print "sys 3072,0: rem to disable"
ID 1080 end
GF 1090 data 208,  11, 165, 253, 141,   2,   3, 165
MG 1100 data 254, 141,   3,   3,  96, 173,   3,   3
HE 1110 data 201,  12, 240,  17, 133, 254, 173,   2
LM 1120 data   3, 133, 253, 169,  38, 141,   2,   3
JA 1130 data 169,  12, 141,   3,   3,  96, 165,  22
EI 1140 data 133, 250, 162,   0, 160,   0, 189,   0
KJ 1150 data   2, 201,  48, 144,   7, 201,  58, 176
DH 1160 data   3, 232, 208, 242, 189,   0,   2, 240
JM 1170 data  22, 201,  32, 240,  15, 133, 252, 200
KG 1180 data 152,  41,   3, 133, 251,  32, 135,  12
EF 1190 data 198, 251,  16, 249, 232, 208, 229,  56
CG 1200 data  32, 240, 255, 169,  19,  32, 210, 255
EC 1210 data 169,  18,  32, 210, 255, 165, 250,  41
AC 1220 data  15,  24, 105, 193,  32, 210, 255, 165
JA 1230 data 250,  74,  74,  74,  74,  24, 105, 193
CC 1240 data  32, 210, 255, 169, 146,  32, 210, 255
BO 1250 data  24,  32, 240, 255, 108, 253,   0, 165
PD 1260 data 252,  24, 101, 250, 133, 250,  96
```

**B128 VERIFIZER** **Elizabeth Deal, Malvern, PA**

```
1 rem save"@0:verifizerb128",8
10 rem* data loader for "verifizer b128" *
20 cs=0
30 bank 15:for i=1024 to 1163:read a:poke i,a
40 cs=cs+a:next i        *
50 if cs<>16828 then print "** data error **": end
60 rem bank 15: sys 1024
70 end
1000 data  76,  14,   4, 165, 251, 141, 130,   2, 165, 252
1010 data 141, 131,   2,  96, 173, 130,   2, 201,  39, 240
1020 data  17, 133, 251, 173, 131,   2, 133, 252, 169,  39
1030 data 141, 130,   2, 169,   4, 141, 131,   2,  96, 165
1040 data   1,  72, 162,   1, 134,   1, 202, 165,  27, 133
1050 data 233,  32, 118,   4, 234, 177, 136, 240,  22, 201
1060 data  32, 240,  15, 133, 235, 232, 138,  41,   3, 133
1070 data 234,  32, 110,   4, 198, 234,  16, 249, 200, 208
1080 data 230, 165, 233,  41,  15,  24, 105, 193, 141,   0
1090 data 208, 165, 233,  74,  74,  74,  74,  24, 105, 193
1100 data 141,   1, 208,  24, 104, 133,   1, 108, 251,   0
1110 data 165, 235,  24, 101, 233, 133, 233,  96, 165, 136
1120 data 164, 137, 133, 133, 132, 134,  32,  38, 186,  24
1130 data  32,  78, 141, 165, 133,  56, 229, 136, 168,  96
1140 data 170, 170, 170, 170
```

# b i t s

*Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits column, we'll credit you in the column and send you a free one-year's subscription to The Transactor*

### C–128 Easy Program Loading

**Martin Hofheinz**
**Stockton, CA**

When saving programs on disk, use the following procedure:

dsave"0:program name<shift space>:

This shows up in the directory as:

"program name":

(the shifted space shows up as a quote). Now you can load a program directly from a directory display – just run the cursor up the left margin of the directory until you get to the program you want to load, press <F2> and <return>, and the program loads without any typing!

You can use the rename command to change the existing programs on the disk to the new shift–space format.

### C–128 80–Column Interrupt Routines

**Stephane Laroche**
**Ste–Foy, Quebec**

Have you ever had trouble programming the 8563 chip (80–column display controller) through an interrupt routine? Due to the way you program this chip, you have to design an interrupt that won't access the 8563 at the same time a program (or the built–in 80–column screen editor) does. A clock that is displayed continuously on the 80–column screen is an example of a program that needs to use such an interrupt.

I found a clean way to resolve this problem. You have to begin your interrupt routine with these instructions:

```
      lda  $0107,x
      cmp #$c0
      bcc  UIRQ
      cmp #$d0
      bcc  HIRQ
UIRQ  jmp  'your interrupt'
HIRQ  jmp  $fa65
```

This will work nicely when you are using BASIC or the monitor. If you are running a machine language program, make sure that every read or write to the 8563 is made through the routines at $CDCC and $CDDA.

### C–128 CAPS–lock q Fix

**Hal V. Weant**
**High Pont, North Carolina**

An undocumented feature of the Commodore 128 is the 'q' key that will not respond to the caps lock key. The problem turns out simply to be the representation in the keyboard lookup table used for the letter q when the caps lock key is depressed. The value of $51 is found where a value of $D1 should be.

The cure is simple, but it will require programming an EPROM. For this reason it is best that you are a technically inclined person who is comfortable with dismantling the C–128 or have someone on hand who is. You will also need to have on hand a Promenade EPROM programmer and a working C–64 or C–128 for burning the chip.

The chip to remove and copy is U35 (next to the socket for the internal function ROM). After the Promenade is made functional on the working computer, copy the EPROM U35 into computer memory with the Promenade command:

£8192,24575,0,5

After that operation, check for an 81 with a peek(23586) to be sure everything is in its proper place. Next, poke 23586 with 209 and then burn a new chip with

π8192,24575,0,5,7

Install the new chip in the U35 socket and reassemble the C–128. You should now find the case of caps lock little q solved.

Note 1: The control word and program method word used with the Promenade are the ones that work for the chips found in most C–128s. Check the Promenade manual to be sure of the proper codes.

Note 2: If your C–128 should have 32k EPROMs instead of 16k ones, adjust the addresses to copy the whole 32k i.e. 8192,40959 and poke 39970,209.

Note 3: EPROMs used are 27128 (16k) or 27256 (32k). 32k chips have not been found in any 128s here, though the schematic shows it to be designed for 16 or 32k chips.

**C–128 Bullet–Proof Windows**

**D.J. Morriss**
**Toronto, Ontario**

The windows on the C–128 have one fault: they're too fragile. You get a nice prompt line all set up, then you accidentally hit HOME twice and the window is "smashed". Here's how to set up a permanent window, at the bottom of the screen.

1. Place the text that you want protected at the bottom of the screen; use any text colours that you like, and use as many lines as you like.
2. Place the cursor a few lines from the top of the screen. Press ESC T to establish a window.
3. Enter POKE 237,23 to protect one line; for each additional line protected, decrease the value poked by one.
4. Clear the window with two HOMEs.

That's it. The bottom line(s) of the screen are now untouchable, even if new windows are set up and collapsed. This works in both 40– and 80–column mode, provided you set up the protected are in both screens. The protected area can be different for the two screens.

**C–128 80–Column Display on a Television**

**Bill Walton**
**St. John, New Brunswick**

When I first purchased my C–128, one of the first things I wanted to try was the CP/M mode. I soon found that I needed the new operating system (the modem support version) if I wanted to get very far with it at all. At the time I did not have the 1902 80–column monitor and was in fact using a small colour television as a monitor. I found that I was in a catch 22 situation. I needed 80 columns to use the program "cpmterm128" (needed to download the new operating system) and it only works in 80–column mode. My solution at the time was not to run out and purchase a 1902 monitor, but to find a workable way to get 80 columns on my TV. I found a rather simple and inexpensive way to do just that and I thought I would pass it along to readers of the Transactor.

All that is needed is three rather inexpensive pieces of hardware, which I have listed below, and a couple of strips of scrap wire to connect it all together.

1. 9–pin D–subminiature male plug (Radio Shack part #276–1537)
2. an RF–modulator (Radio Shack part # 277–221) (an old VIC RF modulator works well, it's what I used)
3. A 5 to 7 volt power supply (Radio Shack part # 273–1454)

You begin the hookup as if you were going to hook your C–128 to a 1702 monitor. First connect two shielded wires to the 9–pin DIN plug. One wire goes to pin #1 or #2 (ground connections), the second wire goes to pin #7, the monochrome output. Run these two wires to the RF modulator, connecting the monochrome output from pin 7 to the video input, and the other line to the ground connection. If you are using a VIC RF modulator, open the case to find out where the connections should go – they are labelled on the circuit board (the ground line connects to the case itself).

After the video and ground lines have been connected, all that remains is to connect power to the RF modulator. The power connections on the Radio Shack RF modulator are fairly straightforward – connect them to to the power supply power outputs, observing polarity. On the VIC RF modulator, the red wire is the "hot" (positive lead), and the case serves as ground (negative).

Lastly, all that is necessary is to plug a shielded cable from the RF output and run it to your TV/computer switch, or directly to the TV antenna terminals. Turn the power on to the RF modulator and turn your computer on in 80–column mode.

For those a little more adventurous, you can tap the needed power directly from the C–128. There are several locations available where the needed 5 volts can come from, including: pin #2 from the user port, pin b–2 from the cassette port, and pin 7 from either joystick port. It should be kept in mind, however, that opening the case to make these connections will void any warranties, and so should not be done on a new machine.

I found that this solution was capable of producing a very good 80–column screen considering the equipment used and the fact you are using a television as an output device. Also I found the hookup was quick and easy to do and required no permanent changes to my computer.

**1541 Half–Track Fix**

**Geoff Rob**
**Victoria, Australia**

Disk read and write errors can arise when the drive gets stuck in between tracks from loading programs that use half tracks as a means of copy protection. Before sending your drive away to be aligned, it is worth running the following short program that steps the head half a track:

```
10 open 15,8,15, "i0"
20 print#15, "m–w"chr$(254)chr$(2)chr$(1)chr$(1)
30 close 15: end
```

Line 20 writes a 1 to PHASE (location $02FE), which is checked during each interrupt by the code in disk ROM at $F99C. If the value of PHASE is 1, then the routine increases PHASE to 2, and steps the head by half a track.

Check by validating a disk. If you get even more errors after running this program, then you can just rerun the program to return to the initial state.

## The Drivelight Zone

Well, I wouldn't have believed it unless it happened to me, and now it has.

I was having trouble with my 8050 drive 0. Loading programs was becoming a real bother, especially long ones. The error LED would flicker, and DOS would occasionally do a head–bump, creating for some rather lengthy waits. Program SAVEs or writing an SEQ file had become impossible on drive 0, but I managed to cope, using drive 1 for nearly everything.

When I figured I could live without my 8050 for a while, I decided it was time to get her serviced. The old sweetheart went almost 7 years without so much as a peek under the hood. I went in to pick her up and was told, "we can't find anything wrong with it". Disregarding the somewhat impersonal reference, I immediately began a process of elimination that lasted all the way home.

"Could I have been mistaken? Was I just imagining all those read errors? No. Maybe I just didn't hit her hard enough at home and whatever chip was loose got jarred on the way in, enough to make her start working again. Maybe my diskettes were bad. No, they worked fine in drive 1. Oh well, forget about it – it's back to normal now and I can get on with my next task."

Or so I thought. I carted her downstairs, set her up, connected the cables, disk in, and Arghgh! It's doing the same thing as before!

I went back to Commodore and watched as they tested it with no apparent problems. So what could it be? "Check your environment", said Roy. Since I didn't have a better idea, that was what I would do.

My 8050 sits on a shelf to the left of a B&W monitor that's connected to one of my C64's. I moved the 8050 as far left as I could, and the monitor as far to the right as possible. Like I said, I wouldn't have believed it unless it happened to me. Just to be sure, I moved the monitor back up to the side of the drive again, and the read errors returned. Turn off the monitor, start a LOAD. . . no problem until the instant I turned the monitor back on.

I don't know when it was that my monitor got too close to the 8050, or vice versa. All I do know is that I wasted two round trips to Commodore, and countless hours dealing with the absence of the sometimes vital drive 0. Except for the aggravation, the experience was worth it and I'm glad I can pass it along. Even though my troubles were with an 8050, I suspect any drive could fall prey to this situation. In fact tape, video, or any unit with a magnetic ingredient to its operation could be affected.

My advice? My advice is not really mine – credit goes to an astute Roy Robinson of Commodore Canada. Even if you haven't moved your equipment, you might save yourself a potentially expensive trip to your service center by simply "checking your environment". Especially if it's read errors from disk. Try moving your drive away from your monitor. Try entering the command, turn your monitor off, and then hit Return. Try changing rooms if you have to. Try your drive with a friend's system if you can. And if you eliminate the trouble, invoking it again is not only excellent reinforcement, but also gives you a feeling of accomplishment. Believe me, I know.

Karl Hildon, Editor–in–Chief

## Obscure C–64 VAL bug

**Nick Sullivan**

The prize for the most obscure bug in Commodore BASIC goes, we think, to a recently–discovered problem with the VAL function. As you know, VAL returns the numeric value of an ASCII numeral string, and most of us have been using it for years with no trouble at all. Yet the code for VAL contains a fundamental flaw that has been lying in wait for the right conditions to manifest itself.

Those conditions were present in the second–last version of the Help! program that appeared in the Transactor, Volume 7 Issue 6. By the time the bug was traced we were nearly ready to go to press; there was time to correct the program, but not to update the article. Hence there is a small but significant discrepancy between the text and the code.

Here is what the relevant part of the VAL routine looks like on the Commodore 64 (if you want to see the whole thing, it starts at $B7AD). By the time we get to the part of the routine listed below, the CHRGET pointer has been saved, and CHRGET now points to the start of the string being evaluated. A pointer to the byte just beyond the string has been set up in locations $24/25. The "JSR $BCF3" at $B7DA invokes a routine that converts a null–terminated string, accessed through CHRGET, to a floating point value in FAC 1:

```
B7CF ldy  #$00
B7D1 lda  ($24),y ;get byte past end of string
B7D3 pha          ;save it
B7D4 tya          ;replace it temporarily
B7D5 sta  ($24),y ; with 0
B7D7 jsr  chrgot  ;lda with first byte of string
B7DA jsr  $bcf3   ;convert string to floating point
B7DD pla          ;recover byte past end of string
B7DE ldy  #$00
B7E0 sta  ($24),y ;put it back where it came from
B7E2 . . .        ;restore chrget pointer and exit
```

Okay, the strategy here is to make the string temporarily null–terminated so that the routine at $BCF3 will know when to stop, then restore the byte beyond the string once the conversion to floating point is done. This approach saves BASIC the extra time and trouble of making a separate null–terminated copy of the string somewhere else before doing the conversion. So do you see the bug?

The Help! program used a 17–byte interrupt routine to switch out the i/o chips and character ROM so that the code in the $D000 RAM could test for a special keypress. This routine could go anywhere in RAM, but its default location was at the top of BASIC, just above string storage. It was protected from being

interfered with by BASIC by lowering the top–of–BASIC pointer by 17 bytes, as described in the article. This seemed like a completely safe thing to do.

And it was, unless you happened to do a VAL on the first string created within a BASIC program. What happened then is that the VAL routine dropped a zero on the first byte of the IRQ routine to null–terminate the string, then went off to convert it. While this was going on an interrupt would happen and the IRQ code, which now began with a 0 (BRK) instead of a $78 (SEI) would be executed. This is a sure–fire crash.

The solution, once the bug was identified, was simple enough: we just lowered the top of BASIC by 18 bytes instead of the 17 needed by the IRQ routine, to give VAL the 1 byte of maneuvering room it required. And that is why the Help! article and the Help! program are not quite in agreement.

## G–Link ROM Compare

**Karl Hildon**

Hey G–Link owners! Have any of you been experiencing the odd glitch with your Glink? Well so have I, and I decided it was time to find out why. And to my complete surprise I found there are at least two versions of the Glink EPROM.

In one version I've noticed that a simple SEQ file read can bomb before reaching the end of the file. With the other I've never seen the problem. I don't know why it happens, just that it does and it's annoying. I also don't know why there *are* two versions, or how it happened. However, once I made all the EPROMs the same, the troubles went away.

On Transactor Disk #18 (for this issue) we'll be including the G–Link ROM file should you wish to re–burn your EPROM. It's a 4K PRG file and it LOADs to $2000.

The program below will tell you if you need to do this update. You might think it looks rather long for such a simple task. Actually, Line 110 alone is enough to determine which version you have. In fact, if you switch your Glink to parallel mode:

print peek(57625) should give 173

If not, you should probably consider doing the update. Remember, I said earlier that there are "at least" two versions – there may be more, but I doubt it. Regardless what value the above PEEK returns, the next part of the program will write the contents of your Glink ROM to a disk file, and then proceed to compare it with the file "glink 3/87 $2000" on Transactor Disk 18. So unless you get Disk 18, there's no point entering the whole program.

Lastly, I've noticed one other glitch, but it very rarely occurs and only with one program that not many will have or use. The utility that creates program listings with the Verifizer codes is called "proofgen". Quite simply, you load a program, LIST it to disk, then SYS to proofgen. The Verifizer code is calculated from the line in memory and written to an output file. Then the same line is input from the disk listing and sent to the output file. Sometimes while proofgen is making this file, the printer will suddenly become the output destination, printing exactly what should be going to disk. The problem here could be in the proofgen program, but it happens so rarely that I can't help suspecting the Glink. If anyone has noticed a similar problem, let us know and we'll go looking for it.

**Glink Compare: RUN in IEEE mode.**

```
LJ  100 rem g–link rom compare
OO  110 if peek(57625)=173 then print "■ your g–link
        rom is probably ok" : goto 150
GN  120 print "■ your g–link rom should be
        re–burned"
JP  130 print "■ with the g–link rom file from
OH  140 print "■ transactor disk #18
IM  150 print
MJ  160 print "if you would like to compare the
FD  170 print "entire rom with the g–link rom
KC  180 print "file on transactor disk 18, this
AN  190 print "next test will write the contents
CL  200 print "of your g–link rom to a disk file
MB  210 print "and proceed to compare the two
GP  220 print "files. we suggest using transactor
HL  230 print "disk 18 so that both files will be
PM  240 print "on the same diskette" : print
AH  250 input "do you wish to continue (y/n)";a$
EM  260 if asc(a$)<>89 then end
FO  270 print : print "creating file 'my glink rom'
PN  280 open 8,8,8,"0:my glink rom,p,w"
LJ  290 print#8,chr$(0)chr$(32); :rem make start
        address $2000
HJ  300 for j=57344 to 61439    :rem g–link range
OL  310 print#8,chr$(peek(j));
KK  320 next : close 8
MH  330 print
LE  340 print "comparing 'glink 3/87 $2000'
HL  350 print "file with 'my glink rom' file
LJ  360 z$=chr$(0)
HJ  370 open 8,8,8,"0:glink 3/87 $2000,p,r
HP  380 open 9,8,6,"0:my glink rom,p,r
OI  390 get#8,a$,a$:get#9,b$,b$ :rem strip start
        addresses
LG  400 for j=57344 to 61439
LN  410 get#8,a$ : a=st : if a$="" then a$=z$
AP  420 get#9,b$ : b=st : if b$="" then b$=z$
HI  430 if a$=b$ then 470
OF  440 x=x+1 : print "■ difference at ";j
IM  450 print "glink 3/87: ";asc(a$);tab(19)"my
        glink: ";asc(b$)
OP  460 print
PK  470 if a or b then 490
EO  480 next
MM  490 close 8 : close 9
JH  500 print "found ";x;" different locations
AB  510 print "glink 3/87 status at end: "a
KD  520 print "my glink   status at end: "b
```

## Input Trickery

**Daniel J. Dyke**
**Bel Air, Maryland**

Here is a C–64 subroutine that I call before any input statement requiring the use of punctuation and the cursor keys:

```
10 gosub 8000
20 input a$
30 print a$
40 goto 10
   . . .
8000 poke 198,5: poke 631,34: poke 632,34
8001 poke 633,157: poke 634,32: poke 635,157
8002 return
```

The effect of this subroutine is that it prints two quotation marks, back spaces one space, erases the second quote with a space, and backspaces again. When the data is entered, the input prompt appears to be followed by a quotation mark. Now any keyboard character, including punctuation marks, can be entered as part of the input data. What is also important to note is that it allows full use of the left and right cursor keys. The computer is not in quote mode, but when the return key is pressed, it thinks it is.

## C–64 Efficient Keyboard–Checking in Assembler

**Ted Beach**
**Arlington, VA**

Quite often we want to suspend a program's execution if the user presses a key. Here is a very simple way to do this. Wherever you need to pause the program, insert these two instructions:

```
waitkey  bit  $c5
         bvc waitkey
```

No registers are used, and only the flags are affected, so the instructions can be inserted almost anywhere. If you want to pause a program until the user presses a key, use these instructions:

```
nokey    bit  $c5
         bvs nokey
```

## C–64 Scrolling Banner Routine

**Andrew Miller**
**Asbestos, Quebec**

While experimenting with the MID$ function on my C–64, I constructed a scrolling "formula" which led to the following program. I find it an attractive attention–grabber, and not bad for 4 lines. After RUNning it and typing in your message, you'll be asked for the viewing area (the number of characters of your message being displayed on the screen at a time). Your message will be automatically centred, and will begin to scroll. Here it is:

```
10 input "message";m$: input "viewing area";va
   : print"■S■   ": b$=m$+"...."
20 a=18-len(m$)/2: if va<len(m$) then a=20-va/2
30 b$=right$(b$,len(b$)-1)+left$(b$,1): c$=left$(b$,va)
   : print"■SQ■   "tab(a)c$
40 for x=1 to 70: next: goto 30
```

## C–64 Undocumented Editing Mode

**Nick Sullivan**

Here's a strange feature in the 64's screen editor. If the high bit of the current character colour is set, like this:

```
poke 646,peek(646) or 128
```

. . .the DEL key will take on a whole new personality in quote mode: if there is any text to the right of the cursor, it will be shifted one character to the left, and the character under the cursor will be deleted. There is one glitch, however: the first time you hit DEL under these conditions, a garbage reverse–character will be printed; subsequent DELs will perform as described.

## Easy Input Speedup from BASIC

**Eddie Anderson**
**Orlando, Florida**

Here is a simple trick that will double your I/O throughput when doing GET#'s from a disk file. Instead of doing a conventional GET# loop like this:

```
4 for j=1 to 80
5 get#5,a$
6 r$=r$+a$
7 next
```

You can almost halve the time required by GET#'ing two bytes with each GET# statements. Like this:

```
4 for j=1 to 40
5 get#5,a$,b$
6 r$=r$+a$+b$
7 next
```

In my testing I read 60 80–byte strings from a 20 block file using each of the above methods. Here is the full listing of the programs:

Version 1

```
1 open 5,8,5,"0:datafile,s,r"
2 for i=1 to 60
3 r$=""
4 for j=1 to 80
5 get#5,a$
6 r$=r$+a$
7 next
8 next
9 close 5
```

Version 2

```
1 open 5,8,5,"0:datafile,s,r"
2 for i=1 to 60
3 r$=""
4 for j=1 to 40
5 get#5,a$,b$
6 r$=r$+a$+b$
7 next
8 next
9 close 5
```

The first version took 95 seconds to finish. The second version took only 49 seconds. I went a couple of steps further to see how much faster the program could be made to run. Changing the program to read 4 bytes at a time made it run in 44 seconds. Version that read 8 and 16 bytes at a time ran in 40 and 36 seconds, respectively.

The times may be off by a second or so because I was timing the runs with a stopwatch. Nevertheless it seems that the biggest gain can be had simply by GET#'ing two bytes instead of one on each GET# statement.

**Systematic Loop–Variable Naming**

**J.G. Krol**
**Anaheim, California**

You can simplify FOR–NEXT loop programming and speed up loop execution time by coding this line before creating any other program variables:

```
110 DIM L1,L2,L3,L4,L5,L6,L7,L8,L9,LA,LB,LC,LD,LE,LF
```

Then as you write the programs: (1) use the lowest–numbered currently–unused L as the loop–variable for each new FOR–NEXT loop you begin writing; (2) don't use these L's for anything else. Whenever there are three active loops, for example, the L1 loop will be outermost, the L3 loop innermost.

This systematic loop–variable naming scheme, though simple, has many advantages over the usual unplanned naming of loop variables.

1. The L–for–loop names are mnemonic, apt, short, clear, and consistent, and they're effortless to invoke when you need them.
2. The systematic naming scheme eliminates the dreaded blunder of reusing an active loop–variable in another loop, thus wrecking the program.
3. The scheme eliminates all the usual head–scratching and "creative" loop–variable naming aimed at avoiding that dreaded blunder.
4. Since unneeded loop–variables are eliminated, the total number of variables is minimized, saving time and space.
5. Loop–speed is maximized since BASIC finds the time–sensitive loop–variables at the front of its variable–table.
6. Loop–speed remains consistent no matter how many (hundreds of) variables you create after the L's.
7. The naming scheme provides a clear, convenient, consistent way of thinking and talking about your program, e.g. "the L5 loop".
8. The systematic names greatly clarify program structure: whenever you spot an L5 in a listing, you instantly know you're inside five active loops – no matter how obscure that critical fact might otherwise be.
9. The systematic names often reveal program structure you weren't aware of. E.g. when you call a subroutine from within an L3 loop, all of its loops must be L4 or higher – and they might not be, since you originally wrote the subroutine you are invoking from an L1 loop. Realizing that, you can easily revise the subroutine.
10. The systematic loop–variable names greatly help other people to read and to grasp your programs, so it's very appropriate for published programs. Contributors, take heed!

When you've completed a program, eliminate unused L's from the DIM in line 110, and reverse the order of the used L's so that BASIC finds the most time–sensitive, inner–loop L's first. For example, if you never used more than the six nested loops (which is typical), the final form would be:

```
DIM L6,L5,L4,L3,L2,L1
```

I've used this systematic loop–variable naming scheme for several years. It's saved me so much time, effort, confusion, frustration, and grief that I wouldn't be without it.

**Selective Scratching**

**Craig Ede**
**Minneapolis, MN**

I recently changed word processors, moving from one which saves files in sequential format to one which normally saves files in program format. My word processing file disks soon became full of a mixture of different filetypes. In an effort to organize things I found out that it is possible to scratch all sequential files from a disk (retaining all other filetypes) by using the following disk command:

```
open 15,8,15: print#15,"s0:*=s": close 15
```

All program files can be scratched using:

```
open 15,8,15: print#15,"s0:*=p": close 15
```

With a DOS wedge of some sort in place, the commands are simpler:

```
@s0:*=s  (for SEQ files) or
@s0:*=p  (for PRG files)
```

(A more selective scratch of files can be done by using a filename pattern other than the '*').

This has proven very convenient in helping me separate the files from my word processors. I make two back–up copies of the file disk and mass–scratch all the program files from one and all the sequential files from the other. It could also be used to delete sequential 'doc' files that are no longer needed on a disk of programs. Though I haven't tested it, I think this will work with user (USR) and relative (REL) files also (check on a back–up disk first).

WARNING: Do **not** use the delete (DEL) filetype with the wildcard as it will scratch all files from your disk. The following is poison:

```
@s0:*=d  (poison: do not use)
```

## Amiga Automatic Up–Date

Andrew N. Mercier, Jr.
Pope AFB, North Carolina

The Amiga provides a very well maintained software clock, and since the Amiga runs a multitasking operating system, the following short routine may be used to keep the time and date on your SYS: disk as current as possible. By writing to the disk once in a while, the time won't be too far off when you re–boot, since the system sets the time to when the disk was used last.

Enter the following command file with an editor or by typing 'copy * to s:saveDate' from the CLI (press CTRL – \ to end your input if you use the copy command):

```
c:wait 720 mins
c:cd sys:
c:date >sys:now
c:run >nil:
c:execute s:saveDate
```

Execute the command file in the background by typing "run execute saveDate". The program will wait 720 minutes, or 12 hours, then write the current date to disk and repeat the whole process over again. The wait uses no CPU time as it runs in the background, since it just waits for a signal from the timer device. If you wish to have the clock updated more or less frequently, change the number of minutes in the first line to whatever you want – every hour might be a reasonable choice.

Note: the CD command is used so that DOS will put up a requester asking for the system disk if it is not in the drive. The redundant use of c: and s: is to ensure that the correct commands are used (in the case of like–named programs residing in the current directory).

## Pattern Matching With The Dir Command

The CLI command "Dir", as documented in the AmigaDOS manual, has quite a few features. An especially valuable capability of this command, however, isn't even documented: pattern matching. Dir will allow an expression for a directory name like those used by the "PAT" option of the List command, and will display all files in the directories that match the given pattern. So, for example, the command 'dir ???' will display the files in all directories whose name is three characters long; 'dir c#?' will display all directories beginning with the letter 'c'; 'dir c|fonts' will display the 'c' and 'fonts' directories.

## Multiple–menu Selections

When using the menus in most applications, you can choose more than one menu option without popping up the menu for each choice. While the mouse menu button is held down and the menu is displayed, move the pointer to the first option you want and click the LEFT mouse button. While the menu button is held down, you can move the pointer to as many options as you wish, clicking the left button over the ones you want to select. When you finally release the menu button, the program will act as if you had selected each menu option normally, one at a time. This is especially handy for menus with choices for many different modes or settings, like a terminal program with options for parity, baud rate, etc. Just click – click – click to set up your choices without having to bring up the menu bar over and over again.

This technique should work with most programs, i.e. those that handle mouse events the way the Intuition manual advises.

## Speeding Up IFF File Access

If you are reading an IFF file from your own program, you can speed up access and shorten the file by removing any chunks that your program doesn't care about. For example, you may be incorporating graphics in a program by creating an image using an IFF–compatible paint program like Deluxe Paint – the picture in IFF–ILBM form can then be read by your program using your own IFF–reading routines, or the standard public domain routines provided by Electronic Arts and listed in the Amiga ROM Kernel manual.

This approach works fine, but there may be information in the picture file that your program doesn't care about, and if you don't plan on using the picture files for anything else, you might as well remove it. In this example, that of an ILBM file saved by Deluxe Paint, The obvious chunks to remove are the CRNG chunks that describe colour–range information, and the GRAB chunk that defines a "hot–spot", or point for grabbing the image. Removing these chunks saves the time needed for the IFF reader to skip over them, which can be considerable on floppy drives due to read/write head seek time.

To take the chunks out, you have to remove them from the file and modify the length of the FORM at the start of the file accordingly. You could write a simple program in the language of your choice to do this, but the easiest way is to use a text editor that lets you edit a binary file as hex. One such editor (perhaps the only one?) is Aedit by Joe Bostic (DRM Programs, 1329 Arthur Ave., Las Vegas, NV 89101). It lets you read a binary file and display it as hex and ASCII. You can edit the hex values, and they will be converted back to binary when the file is saved. Using Aedit, it is a simple matter to remove all the CRNG chunks and the GRAB chunk, leaving just the initial FORM, the ILBM, BMHD, and BODY. The longword after the initial FORM will have to modified to reflect the new size of the file. For simple single–picture IFF files, the FORM length should be eight bytes less than the size of the file as it appears in your directory (using the CLI List command).

Minimizing your IFF files like this can yield dramatic reductions in reading time, especially for small files.

# Letters

**Wanted – Plus/4 Technical Info:** Can you or a reader help me? I want a memory map for the 6529 chip in the Plus/4. One like the excellent Figure C–10 in Appendix C of Jim Butterfield's fine book *Machine Language for Commodore Machines* would be ideal.

Also, where and how can one get a copy of the circuit diagram of the Plus/4? That would be a big help in understanding the machine and in machine language programming.

Jim Welch, Santa Clara, California

**Wanted – an IMG stripper for tape:** Congratulations on your fine magazine, which I have just discovered. "Downloading From CompuServe" by Christopher Dunn (Volume 7, Issue 4), has certainly enlightened me with a problem I have been encountering.

I have been downloading with an XMODEM protocol terminal program on tape. The reason why the IMG files have refused to run is now clear.

Would one of your technical staff, a contributor or reader be able to help me? I would like to obtain a copy on tape of an XMODEM protocol terminal program for the Commodore 64 capable of downloading both BIN and IMG files with an IMG byte–stripper built into the program allowing IMG programs to run. Alternatively, a listing of same would be most appreciated.

My address is: 17 Sugarloaf Drive, Chirnside Park, Victoria 3116, Australia.

Philip Gahan

*We took a stab at your dilemma, Philip, but quickly determined that without a sample copy of a tape, it wouldn't be wise to print a program that strips an IMG file header based on speculation. If you hear from someone that already has this program, then I guess we don't need to re-invent the wheel. If you don't, send us a tape with a .BIN file on one side, and a .IMG file on the other, and we'll take another shot at it.*

**A lead on cables:** A possible supplier for letter writer Doug Hurd of Penticton, British Columbia (Letters, Volume 7, Issue 6), who needed right angle connectors and long serial cables among other things, is Value Soft, of 9513 S.W. Barbur Boulevard M–56, Portland, Oregon 97219. They have an ad in Run magazine (3/87) listing a 9–foot right angle serial cable for $9.95 (US).

Keep up the great work. The Transactor with the TPUG insert has got to be the best deal anywhere for Commodore freaks! Really enjoy the cover art. Volume 7, Issue 6, with the robot sculpting a self–portrait, is perfect for an issue with an article on recursion.

You, or your readers interested in recursion and AI, might want to look at Douglas Hofstadter's books: *Godel, Escher, Bach* and *Metamagical Themas*.

John E. Graham, Director EMC, Sinclair College, Dayton, Ohio

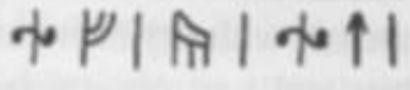
*Thanks for the lead, John. And we'd also like to endorse your recommendation of the Hofstadter books. One of the things that distinguishes Hofstadter's AI work from most others is its emphasis on the "intelligence" in AI; he's interested in intelligent machines, not just clever programs.*

**Another IEEE for the 128:** On page 14 of the March issue, in a reply to a letter from Bill Bennett, there was a request for information about a good IEEE interface for the C–128. The unit I am using is an INTERPOD from England. I have been using this unit, first with a C–64, and now with the C–128, without troubles for the past three years. I use the INTERPOD with a four–pole double–throw switch so I can go between the IEEE disk drives and the MSD–2 and 1571 drives when using the C–128.

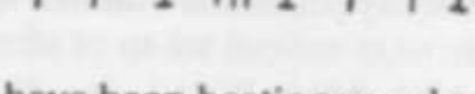
I do not know if the unit is still available. I wrote a letter to the address in England about ordering another unit, but never received a reply. I purchased the unit from: Oxford Computer Systems (Software) Ltd., Hensington Road, Woodstock, Oxford, England OX7 1JR.

John J. Schueler, K7QV, Sedona, Arizona

**Ultimate frustration fix:** After months and months of playing Ultima IV (six to be exact), I finally made it into the Abyss. But when I got there and had to answer the *very last question in the whole game*, I got it wrong. It took me a couple of more months to figure out what was wrong: there is an error in the visions you get when praying in the shrines. Instead of getting:



you should get the following:



Now all of you who have been beating your brains out trying to figure this out (as I was) can finally get on to your destiny. . .

Paul Reeves, Hamilton, Ontario

*Ordinarily we don't publish game tips, and this isn't meant to be a precedent (so put down your pens right now). But the idea of Transactor readers beating their brains out for want of this information was more than we could stand.*

**Biblical Greek drills:** If any of your readers would want some drills on Biblical Greek, have them send me a disk with something they think is really useful, educational, or entertaining, and in exchange I will send them a disk with the Greek on it. These are more than vocabulary drills. Included are exercises on identifying verbs, nouns (5 case system), participles, infinitives, and adjectives on the book of I John. II Thessalonians is the next book of the New Testament that will be covered. My address is: 2311 Creswell Road, Bel Air, Maryland 21014.

Daniel J. Dyke

**Hardware book blurb:** I would like to inform you readers of a book I have written which I believe may be of interest to them. Entitled *Electronic Computer Projects*, it is published by COMPUTE! Publications Inc., and is now available.

*Electronic Computer Projects* is an introduction to computer interfacing and digital electronics. It is written for users of the Commodore 64, VIC 20 and C–128, as well as the eight–bit Atari computers.

As a computer and electronics hobbyist, I believe this book will be helpful to users who have done some exploring of their machines programming–wise and are looking for new, interesting and exciting things to do with their computers. This book introduces the reader to computer control and monitoring of items in "the outside world" through the construction of projects consisting of a hardware and a software component. While the projects are useful in themselves, they also provide a basis for readers to begin designing circuits for projects of their own.

The book is available from: COMPUTE! Books, P.O. Box 5038, FDR Station, New York, New York 10150 (toll free 1–800–346–6767; in New York, 212–887–8525); or in Canada from: McGraw–Hill Ryerson, Ltd., 330 Progress Avenue, Scarborough, Ontario M1P 2Z5. The suggested retail price is $9.95 (US).

Soori Sivakumaran, Burlington, Ontario

**Grounding to a halt:** Your issue on Gadgets and Gizmos inspired this letter. I am an amateur radio operator, and use my C–64 mostly in connection with my radio hobby. Some months ago I lost a 6526 chip and had quite a bit of trouble getting another from Commodore. I would like to caution your readers who are interconnecting their computers with other equipment about one of the sneaky problems that can spoil their fun.

In my case, I have my 64 on the bench with my amateur radio equipment and general test equipment. For safety's sake, one learns early to ground everything, so I have bare copper wires running across the bench at several spots where they are out of the way but easy to connect. All equipment is of course grounded. But originally not the Commodore, of course. It has a 1541, an 801, and a monitor, all interconnected. And each plug has a ground; what more could be needed?

When I bought the monitor, my wife advised me to buy a colour TV instead of a Commodore monitor, since we could also use it for another TV if the need arose (not to mention how I would use a computer without a monitor!). Since she has been advising me wisely for thirty years (and I know where my buttered bread comes from) I bought the colour TV, a nice RCA 13 inch colour set. And it did well. When I decided to connect my Commodore as a terminal for my RTTY interface, I was afraid of damaging the CIA chip, so I carefully studied the diagrams, and finally decided that a buffer was called for, to be absolutely safe. So I mounted a couple of 7404 TTL chips on a perf board, and with an edge connector on one side and a socket on the other it fit perfectly on the USER interface, and was powered from the Commodore. With diode clamping to TTL levels, and with the Commodore chip a CMOS (and able to take 15 vdc), *nothing* could get through that buffer and hurt my computer.

Or so it seemed. I connected the cable from the RTTY interface to the buffer, turned the system on, and promptly burnt out two inputs on the CIA. The computer would still work for programs and games, but no more CompuServe!

It's a good thing, although I didn't think so at the time, that it took two months to get a chip. I was sure the buffer had failed me. After all, electronic equipment is untrustworthy in general, and sometimes tricky. Using analyzers, oscilloscopes, and every technique I have learned over the years, I hunted for the elusive intermittent that had damaged my baby Commodore. Nothing. After a month of part–time testing I gave up. Another month and the new chip came and I had had time to use my last resort technique. I thought about it. Then I connected a voltmeter between the RTTY and the buffer input (with no other connections) and turned on the radio. No potential difference. I then turned on the Commodore, the 1541, the 801, and the TV. When I switched on the TV, the voltmeter immediately jumped up to 30 vdc. And an old memory surfaced. Solid state tuners sometimes put out a dc voltage, which hurts them not at all, and is not noticeable, but which is enough to raise the ground on the Commodore to the same voltage. Then if connected to other equipment which is truly grounded, you have a problem. The fix is simple. Be sure your Commodore system, when powered up, has no difference in ground potential with the system you are connecting into. Otherwise, you may, like me, be waiting for a chip.

I enjoy your magazine generally, although many times your writers assume I know more about a subject than I do, and then I am lost and cannot follow what they are doing. Keep up the good work. You have the best magazine on the market.

Carl L. Henry, Bowling Green, Kentucky

**Holes in Inner Space:** I would like to take this opportunity to thank you and your excellent staff for producing what is quite certainly the most useful book currently in print for Commodore computers: *The Complete Commodore Inner Space Anthology*.

Thanks for the COMAL memory maps; those are the only COMAL maps I've ever seen. And yes, I even use the periodic

tables! The Wordprocessing Reference Guide was also a wonderful idea – even slightly more accurate than the PaperCLip manual. I trust Speedscript commands will be included in Anthology II as well.

As I am sure you already know, CCISA is not entirely complete. In Anthology II, I would appreciate a small section detailing physics formulae. The current small section on forces is really only a start.

I make regular use of both the unit conversion tables and the geometric volume tables; however, I noticed the omission of the universal volume equation from these fine tables:

$$V = (T + 4M + B) \times h / 6$$

T = Area at top
M = Area at midpoint
B = Area at base
h = height

This equation is particularly useful when dealing with irregularly shaped figures. (In case one should ever happen to forget how to calculate the volume of a frustum of a cone.)

Kevin Smith, Edmonton, Alberta

*Thanks for the input, Kevin. As you know, we try not to leave anything out. Perhaps you could send us a list of those physics formulae you mention, or maybe just a list of the results they produce so we could research them for you. And if anyone else has ideas for Anthology type info, we're always open for suggestions.*

**Structuring Basic programs:** I read with interest the article "Structured Programming on the Commodore 64" by Frank DiGioia (Volume 7, Issue 5). I program on both large mainframe computers (for a living) and small micros (for fun) and try to use the techniques of top–down design and modular programming in both environments.

While the extra structures (WHILE–WEND, REPEAT–UNTIL, CALL–PROC) can help your program be more structured, it is certainly possible to write structured Basic with just Basic 2.0 on the C–64. Top–down design is possible in most programming languages, and Basic is no exception. Modular programming is also possible providing we follow the cardinal rule of "one entry, one exit". This rule is what has given the GOTO a bad name. If we only use the GOTO to branch to the RETURN of a GOSUB or the NEXT of a FOR–NEXT (perhaps with a "CTR = MAX" statement to make our FOR CTR = 1 TO MAX loop end nicely) we are in great shape. The only other thing to watch for is not entering a GOSUB–RETURN module except for one place: the top.

If we need to go to one of several places after exiting a module (processing one of several types of input records read by a common read routine for example) a "return code" can be set by the module to a value allowing an ON. . .GOSUB (not GOTO, please!) to direct us. This is the only problem I had with Mr. DiGioia's otherwise excellent article. Perhaps the EXIT nn keyword could be modified to set a pseudo–variable (like ST) with the value of "nn" as well as exiting the structure normally. In this way we could know that we would always come back to the point we left when doing a CALL or GOSUB and still have the flexibility to decide what to do when we get back.

Douglas Hattrem, Lansing, Michigan

**FOG clarifies:** The March 1987 issue contains an article entitled "Compatibility and Operability of the C–128 CP/M+ Operating System" by Ralph Morrill. As the leading international user group providing support for CP/M, we at FOG were naturally pleased to see the vastly under–explored CP/M+ side of the C–128 being brought to light. Thousands of C–128 owners have joined our group seeking just the type of information Mr. Morrill writes about.

One point of the article needs explanation. Mr. Morrill writes, "Not one piece of Osborne software from the First Osborne Group's (FOG) public domain library that I have acquired will run on the C–128." As a matter of fact, the vast majority of our *current* library runs on the C–128, as attested to by our growing C–128 membership. FOG's library is tested, documented and well organized. Programs which are computer–specific are clearly labelled.

Our library was once divided into computer–specific sections, including libraries for Televideo, Kaypro, Morrow and Osborne. It has come to our attention that disks of our old library are being sold by some public domain software houses. Please warn your readers (and writers) that the reliability of the software acquired from many of these sources is at best questionable. It is clear that Mr. Morrill did not acquire his software from FOG.

Another possible source of Mr. Morrill's problem may be that the disks he tried were Osborne single density format. Our current library is on double density format disks. The 1571 drive will read Osborne format only if it is double density.

I really wish Mr. Morrill had called our office when he had a problem. FOG has a full–time technical support staff to help our members. We certainly would have made sure Mr. Morrill had up to date information for his article.

FOG welcomes C–128 owners to join our ranks. Our award–winning FOGHORN newsletter is unmatched as a source of CP/M information. Our initial offer to C–128 owners of a free disk of CP/M software with membership has been such a tremendous success that we have extended the offer through December. Your readers may write to us for further information at P.O. Box 3474, Daly City, California 94015–0474, USA. FOG is a non–profit corporation.

Ronald V. Forsythe, FOG President, Daly City, California

*Thank you Mr. Forsythe! We've been waiting for a letter like yours a long time. I'm sure many of our readers will take you up on your offer, especially the ones that have previously inquired about just such a source of CP/M information.*

**Simplifying the raid:** Say again? I use relative files a lot, and that occasional bug drives me crazy. I welcome Shiloh's Raid (January 1987) but I don't want to run an eight hour test, I just want the files to work. I've spent 2 hours studying the article and just want something simple. Can you print just a small routine to be inserted when positioning to a record?

Max Chapman, Reno, Nevada

*Well, maybe it was a little cryptic, Max. However, if our calculations are correct, the subroutine below should do the job for you. When you open your relative file, set the variables LF and SA to your file's logical file number and secondary address respectively, and set the variable L to the record length of the file. You should also initialize to zero the variables SR, R and W, which are used internally by the subroutine. Before writing a relative record, just set the variable N to the record number you want, and call the routine with GOSUB 9000. With this version you will have to write the whole record; positioning to mid-record is not supported.*

```
8970 rem  position to relative record before write
8980 rem  from shiloh's raid (c) 1987 david shiloh
8990 rem  adapted from the transactor, volume 7 issue 4
9000 if sr then r1 = sr + 1: r2 = sr + 2: r = 2
9010 q = n*l: q% = q/254: q = q-q%*254: sr = -q%*(l>q)
9020 if sr then sr = -q%*(q-l-1)
9030 h% = n/256: lo = n-h%*256
9040 rem point twice and wait if needed
9050 if r then r = r-1: if q% = r1 or q% = r2 then gosub
9060: w = 162
9060 print#lf, "p" chr$(sa)chr$(lo)chr$(h%)chr$(0)
9070 if w then poke w,2: wait w,32: w = 0
9080 return
```

**Double Density HR S&P:** I wish to express my thanks to Mr. Bose (Letters, Volume 7, Issue 5) for his kind comments on Hires Search and Print. His suggestion to enhance the program's output with double density graphics is a good one and I have taken it. Six bytes must be altered and eight more added to the program to effect the change. This is a simple fix in the source code, and I include the information here for those who entered the program this way.

Under CKMOR: tell program to send 800 instead of 400 bytes for large print

```
LDA #32          replaces   LDA #144
STA CODE + 2     replaces   STA CODE + 2
LDA #3           replaces   LDA #1
```

Under END: reset single width codes for 400 rather than 200 bytes

```
LDA #1           (added code)
STA CODE + 3     replaces   STA CODE + 3
LDA #144         replaces   LDA #200
```

Under NXTBYT: print each byte one more time to compensate for double density. Note: this change is required in two places within NXTBYT (ie. replace the two single "JSR CHROUT"s with double "JSR CHROUT"s)

```
JSR CHROUT
JSR CHROUT      replaces   JSR CHROUT
```

Under CODE: change printer codes from 200 to 400 bytes per line

```
DFB 27,76,144,1 replaces   DFB 27,75,200,0
```

Unfortunately, no easy change is available for those using the Basic loader. Quite a few DATA numbers change after re-assembly due to address alterations caused by enlarging the program. For those, I have rewritten the Basic program to include the enhancement and have formatted it as a generator rather than a loader. This will create a disk-based object file that loads much quicker and takes less disk space. If there's room, perhaps it can be included on the The Transactor Disk in a future issue.

Jack R. Farrah, Cincinnati, Ohio

*Thanks for taking the trouble, Jack. Though there is unfortunately not enough room in the magazine to print the new program generator, the double density version of Hi-Res Search and Print will indeed appear on Transactor Disk #18 (this issue).*

**"Fast File" relocation:** The "Fast File" program (Volume 7, Issue 4) on page 6 contains an absolute jump instruction (JMP $C050) and is therefore *not* relocatable as written.

The unconditional jump can be transformed to a relative jump by replacement with:

```
CLV
BVC +7
```

where "+7" is the positive displacement of the relative branch instruction. Make these changes in the original program:

```
60 if c<>11584 then print "!data error!": stop
180 data    8,  76, 249, 224, 132,  97, 184,  80
190 data    7, 145,  53, 200, 196, 252, 144, 232
```

It would be nice if Rick Nash furnished more information concerning the undocumented parts of the SYS instruction used in the program, along with the source code to accompany the machine language data statements.

I would like to see if it is possible to modify the "Fast File" utility for use with COMAL 0.14.

Lewis C. Brown, Rowayton, Connecticut

**A dissenting vote, for C Power:** I am a programmer, experienced in many languages. One of my favourite languages is C. I own a Commodore 64 and enjoy programming it because everything is so accessible with so little effort! The introduction of C compilers for the C-64 was a great day for me. Naturally I

read with some interest Adam Herst's comparison of C compilers in Volume 7, Issue 5, and I feel I must take issue with some of his conclusions. I purchased both of the compilers when they were introduced, and quickly decided on the superior package – C Power. Mr. Herst decries several alleged shortcomings of C Power, and ultimately recommends Super C for novice to intermediate programmers, and says that it is a toss–up for experienced programmers. This is unfortunate. I would like to address his conclusions individually and provide my own observations.

First, there is the question of documentation. Mr. Herst lamented the lack of a tutorial in the C Power package, and the dearth of examples. I found that there were sufficient examples of everything, especially in light of the clear explanations given in the documentation. Most compilers that I have purchased do not come with language tutorials. A compiler is a tool. It should contain instructions on how to use it. A tutorial is extra. (Word processors don't contain lessons on grammar, do they?) I found the documentation that accompanied C Power to be quite complete and accurate. It is to Pro–Line's credit that the documentation was clear and short. It seems to have been written for programmers, as it should be. I am able to find answers to all my questions quickly and the answers are matter–of–fact, not tutorial in style. I like that. My copy of C Power came with the *C Primer Plus* book from SAMS. This is an excellent text, and I find that I use it as my primary C reference. Pro–Line did indeed drop the text in later releases but, as I said before, it was extra to begin with. I did find the C tutorial in the Super C package to be excellent. Indeed, I thought it was the best part of the package, but to condemn the C Power package because it does not contain a tutorial is missing the mark.

The lack of graphics functions in C Power is unfortunate, but they are very easy to implement. Otherwise the library functions are complete for the vast majority of programmers.

The linker options in C Power are much more flexible than those provided in Super C. You can link your C programs to be loaded and run as a Basic file. You can also link them to be loaded and run under the shell, with command line arguments and i/o redirection. You can also link them to a fixed address to be initiated with a SYS command (for wedges and other nice things). Super C programs must be executed from the Super C shell.

The one area that is most important to many programmers is execution speed. He does mention that C Power has the edge in that category. HAS THE EDGE?????!!!! My version of the prime number sieve program took 11 times as long with Super C compared to C Power. Eleven minutes compared to one minute. Super C code executes slower than compiled Basic. The one benchmark program that he gives results for is i/o intensive (a file conversion program). That is not a valid test of program execution speed. The 1541 disk drive is the slowest drive on the market and the 1571 is not much better. His program is limited by disk speed, not program execution speed.

Last, let me comment on the overall feel of the package. The C Power shell is UNIX–like, in that it is command–line driven, allows i/o redirection, and some standard UNIX utilities are included. I like the command interface. I find it straightforward and functional. Indeed, functional is my description of the overall package. It is not flashy, it is not cute, it simply does what it says it will do. That is extremely important when writing programs, as it allows me to concentrate on the program being written, not on the environment in which I am writing.

In conclusion, I can only recommend the C Power package. The non–standard format of the text files used by Super C, the fact that programs can only be run from the shell, and the slow execution times make it the less desirable of the two packages.

Herb Rose, Dale City, Virginia

**C Converter for Super–C v1:** I was happy to see Adam Herst's article on C for the C–64. I was beginning to wonder if I was the only C–64 owner working with the language.

Unfortunately, I am using Super–C Version 1. Adam's file converter will not compile and link as written with that version.

The changes necessary to properly compile and link the program with Super–C v1 are as follows:

In main( )
1) #include "a:stdio.h" **to** #include "stdio.c"
2) fgets(inbuff,254,inchan); **to** gets(inbuff,254,inchan);
3) status = EOF; **to** status = EOI;
4) fputs(outbuff, outchan); **to** puts(outbuff, outchan);

In namefile( )
5) gets(name,16); **to** gets(name,16,STDIO);

In convert( ) and copybuff( )
6) char inbuff; **to** char *inbuff;
7) char outbuff; **to** char *outbuff;

Changes 6 and 7 occur because Super–C v1 defines undimensioned arrays as having null elements, except where initial values are set at definition. In the case of the Convert program, moving elements between the null arrays produces an overflow error. Using pointers remedies the problem.

Changes 1 through 5 are due to differences in the standard library.

Thanks for a fine magazine.

Terry D. Decker, Oklahoma City, Oklahoma

**Super C and the RAM disk:** In reference to Adam Herst's article, "A Tale of Two Cs" (Volume 7, Issue 5): I have taken the liberty of modifying his "Seq/Usr convert" program (for Super C) to allow it to be used from the A or B disk drives or the RAM (H) disk. I also deleted the printing portion of the program as I found this saved a lot of time in the conversions.

Converting a 30 block file I got the following times:

| | |
|---|---|
| original (a: disk to a: disk) | |
| with file printing | 1:21 |
| new (a: or b: to a: or b:) | |
| no file print | 0:46 |
| new (a: or b: to ram disk) | 0:25 |
| new (ram disk to a: or b:) | 0:37 |
| new (ram disk to ram disk) | 0:18 |

I feel that these times make it worth the trouble to copy the program to be converted to the RAM disk.

In regard to the RAM disk, I contacted Abacus Software and they provided me with a program to increase the use of expansion RAM to 128K. After experimenting with it for a while, I finally found out how to expand the RAM disk to 1985 blocks or 496.25K, using the following program:

```
1 #include "h:stdio.h"
2 file outchan;
3
4 main( )
5 {
6    *(char *)0xed06 = 0x7e;
7
8    outchan = open(15, 15, "u:");
9    close (outchan);
10 }
```

The following hex numbers poked into $ed06 by line six of that program results in these configurations of the RAM disk:

| | |
|---|---|
| $30 | 737 blocks free (the expansion suggested by Abacus) |
| $40 | 993 blocks free |
| $60 | 1505 blocks free |
| $7e | 1985 blocks free |
| $7f | 0 blocks free |

After $7f the count seems to start back up again. It is interesting to note that although at $7f the blocks free were shown to be zero, nothing in the current RAM disk was erased.

So much for the good news. We must now deal (again) with copy protection. When I tried to copy the Super C editor, compiler and linker to my big new RAM disk, I found I could not. When I asked Abacus why, I was told because it was copy protected. . . They said there was nothing they could do about it. . .

Well, in my other C-128 mode, CP/M, a friend loaned me his MIX C Compiler to try. I went out and bought it although I could have just copied it, as it was not copy protected. . . It would of course not have been much use without the very large manual that comes with the program. Super C would be even less use without its manual because of its non-standard features. So I guess the thing to do now is go out and spend some more money on a program breaker that will allow me to use the program I have already purchased in the way it should be used.

Or I can use MIX (which, by the way, is a very good implementation of C), which works beautifully on the RAM disk in CP/M.

I will now get off the soap box. I hope all of this has been of some interest to you and the rest of your readers.

Joe Faust, Lompoc, California

*Unfortunately, we don't have space here for Joe's revision of the Convert program. However, it will be included on the Transactor disk for this issue.*

**TransBASIC graphing on one screen:** The TransBASIC graphing module by Paul Adams is the answer to a prayer – almost. The program I want to use it in is so large that I just can't afford the luxury of two hi-res screens. Is it possible to modify the graphics routines so that they only write to one screen and leave the other 9k available to Basic?

I don't have any idea how big a project that would be. I hope someone will take it on, and I hope you will be inclined to print the changes in an early issue.

The Transactor is great. Please keep on doing what you're doing.

E.D. Berners, South Bend, Indiana

*It looks from the code as though the simplest way to make the change you want involves altering exactly four lines of code in the original module. This is going to leave you with a module that is much larger than necessary (it will still contain all the now-redundant code to handle switching between the screens), but a wholesale rewrite is unfortunately beyond our resources. Here are the modifications you would have to make:*

```
13002 ; sta (t4),y
13038 ldx #3
13502 f400 .byte $8f,$48,$00,$00,$00
15146 hite .word $e000
```

*We should warn you that this set of changes is untested; however, we think they'll do the job for you. Good luck.*

**Bundling TransBASIC dialects:** I would like to get some instructions for the use of TransBASIC, if I could. I ordered the package a few months ago from the Transactor, and have found it a worthwhile, cost-effective program. However, I do not understand how to link my own programs to the TransBASIC dialect that I have used to write the program.

First of all, I must explain that I am not a hacker or a "techie"; I am a high school English and history teacher and am interested in writing some simple software for use in my classes. I have found Basic 2.0 to be cumbersome, so I have been looking for a Basic extension that I can use with my program which would not have to be loaded prior to loading my program.

Is it possible to do this with TransBASIC? If so, could you explain to me in simple, layman's terms how to do it? I have read the manual several times and, as far as I can figure, I have to use the Datafier to make data statements. Beyond that, I have no idea what to do.

Bob Irving, Brockville, Ontario

*After you have assembled your TransBASIC dialect, and noted the hexadecimal start and end addresses of the object code as given by the assembler, you should load and run the Datafier. After the run, your dialect will be in memory in the form of a loader subroutine beginning at line 9000 (you can renumber it if it will conflict with the Basic code you have written).*

*The next step is to merge this subroutine with your Basic program. First save it to disk as a temporary file (e.g. SAVE "TBTEMP",8). The easiest way to do the merge is with the ADD fast merge command on the TransBASIC disk. You can enable this command by loading and running the TransBASIC program again as though you were about to create a new dialect. Now load your Basic program and type, ADD "TBTEMP". When you get your READY prompt back you should have both your original program and the Datafied dialect in memory as one program. All that remains is to add the following lines to the beginning of the program:*

```
1 if peek(773)<192 goto 3
2 exit
3 gosub 9000
```

*This will cause the custom dialect to be poked into memory and enabled on each run of the program. Save the modified program back to disk and you're done.*

**Amiga music software shortage:** I am a professional composer and musician, and the proud owner of an Amiga since October, 1985.

When I purchased this computer, it was on the basis of the immense promise it held. The combination of power, expandability, and ease of operation in a multitasking environment, with standardized storage of graphics, text and sound files to accommodate transfer between programs made the Amiga my obvious choice.

Oh sure, I knew there would be a bit of a wait for the software. Programs like the complete rewrite of Roger Powell's "Texture" sequencer or the numerous other music packages from Cherry Lane Technologies might not arrive for a few months yet. . . but the wait was bound to be worth it.

Here I am, well over a year down the road. What have I got to show for my patience? Well, there's Activision's "The Music Studio". Although not bad as a music toy, its laborious system for entry of notes (the Amiga version doesn't even allow you to use midi for this) makes their boast of it being "a professional–quality music composition tool" seem strange. Electronic Arts' "Deluxe Music Construction Set" comes much closer to fulfilling that promise. The system it uses for placing notes on the staff is a lot more flexible. . . although it seems odd to me that they wouldn't use the even more elegant method established in the Macintosh version (where you place the note on the score, and then drag to the right to choose the timing value – what could be simpler?)

I have the Mimetics "Soundscape Pro Midi Studio". As a multitasking work tool, this could be a big help to me some day. . . but at this point the sequencer section – which is what I have the most need of – is lacking in any of the sophisticated editing features most other sequencers coming on the market have. Still, I remain optimistic on this one.

As an example of a clever music program which does the sort of ground–breaking things people were predicting for the Amiga, all I have at this point is EA's "Instant Music". This is a wonderfully entertaining program, loads of fun at a party. Although not really a composer's tool. I still applaud the authors for a job well done. I wish there were more music programs as imaginative.

And while I'm wishing, here are some other wishes. Why is the Amiga the only major computer on the market today to not have:

- a full–featured midi sequencer
- a voice librarian, at least for the DX7, preferably for others, and
- a visual editing program for the Mirage.

Nothing would please me more than to hear that I am wrong, and that programs like these are available already. However, after many months of perusing the various computer and music related magazines with not so much as a rumour, it would be quite a surprise. So how about it. Does anybody have any surprises?

Rob Bryanton, Regina, Saskatchewan

**More Superkit sorrows:** I would like to add yet another voice to that of Roger Detaille, Therese, Quebec (Letters, Volume 7, Issue 5). I also purchased a copy of Superkit 1541 during the summer of 1986, after reading your glowing review (your review is still 'glowing'!) I suffered some real problems with this program. Many of the functions did not work correctly. I have written two letters to Prism in Waco, and have never received any acknowledgement that I had even written! In short, "little boy, don't bother us", seemed to be their attitude. I realize that the program only cost $29.95, but one should receive some value for the money. I also realize that I am a 'hotdog eater' subscriber to Transactor. I notice that you have accepted advertising from Prism and suppose that that will make all reviews sound very good! I have written them twice, even about whether the new version, which I am supposed to be able to purchase for $10, was a significant improvement over the first version; complete silence!

Another comment: the reply which you gave Mr. Detaille was the worst "baloney" I've read in a long time! The line, "we don't think you'll be disappointed with the quality of Prism's after–

sales support to registered owners" is really off beam. They don't even answer letters to their "registered owners".

Another comment: on page 17, you seem to be orgasmic about the fact that CompuServe lets you upload without fee. Yes, but all those uploaded programs are downloaded at CompuServe's regular on–line charges, and become a source of 'free' materials which they, in turn, sell at a profit. Why don't you tell the whole story?

You might think that I am a dissatisfied customer of Transactor, but I'm not! I love the magazine and would like to see it become (if it's not already) the ultimate Commodore mag in both countries. I have dropped my subscriptions to some other "junk" Commodore mags, subscribe to COMAL Today & Disks and might try a Transactor Disk Subscription soon. I was upset about TPUG's handling of my subscription (6 months of no communication) but hopefully that is resolved now. Keep up the good work and take a closer look at SuperKit 1541. I have used FastHackem and DiSector, and both have done a better job for me on the same system. My two drives can't be that far out if I have never had any problem; the alignment program also says so!

Robert L. Manchester, HDE, East Aurora, New York

*That was a welcome change of pace in your last paragraph, Robert. We're sorry you've had trouble with SuperKit and with Prism, but anything you've read about them in this magazine has been strictly in good faith. It doesn't surprise us when people have the odd problem with the SuperKit program – it is definitely sensitive to bad alignment, and probably to discrepancies in rotation speed, as well. But on a good drive, our experience is that it works as advertised. As for Prism's support of users, most of what we hear is positive. But we don't have any institutional relationship with Prism, and no inside knowledge of how they treat their customers. (By the way, what is a "hotdog eater subscriber"?)*

*If somebody at Prism would like to respond to Robert's letter, we'd be glad to print that response, and interested to hear why his letters have gone unanswered. You there, Mr. Domengeaux?*

*As for our not telling "the whole story" about CompuServe: you'll have to excuse us for not realizing that CompuServe charging for downloading time is a newsworthy item. At one time,* **uploading** *programs to CompuServe was chargeable online time – and submissions were frequent.*

*On reflection, we now realize that, by not making a point of this, we were being unfair to all those other world–wide online services who operate as public charities. Sorry, guys.*

## TransBloopurz

### Cap Meter

Norm Herman of Oregon City, Oregon, writes: "The "Cap Meter" schematic in your January 1987 issue worked perfectly. . . There was a small problem with the program, however. The calculation throughout was superb, but since two ranges of values calculated out in scientific notation, both employing the E, and since the ASC(P$) is 69 for both ranges, both came up as PF. The E–03 had to be split off, which can be done by adding a line 565, thus:

```
565 if p$ = "e–03" then print ".00" x$ " uf": return
```

### Bits

Shirley Karish of Hicksville, NY noted a slight error in last issue's bits section: "in "Printer output from an ML monitor", the very first line of the program should be LD**X** #$04 instead of LD**A** #$04. With that little change, she reports, it works just fine."

### N–Body Simulator on Transactor Disk 17

The "N-Body Simulator" program was fine as listed in the magazine, but the program on the Transactor Disk (disk 17) had a few problems: lines 220 and 2350 are missing and prevent the program from running. To correct the program, just LOAD it and enter these lines from the listing on page 33 of Volume 7, Issue 6.

### Some Notes on Transactor Disk 17

There are two PRG files on Disk 17 that have raised a couple of questions. "ATOM" is loaded by "REVERSI", and "ROCKET" is loaded by "FENCE". Loading ATOM or ROCKET themselves is meaningless.

The N-Body Simulator program was submitted with ready-to-load files that would've saved you from typing in the values from the charts. We regret not including these on Disk 17, so we've decided to put them on Disk 18, along with some extra systems that we've come up with ourselves. The files can be identified by the ".NB" suffix, but don't enter the suffix when specifying the filename for a load. For convenience we'll put another copy of the simulator on Disk 18 too.

### A Two-Button Mouse

Charles McCarthy from St. Paul MN, wrote with the following correction: "This concerns the nifty article about the C-1350 mouse (volume 7, issue 6, pages 36-38). There is an error in the code for the procedure TEST on page 38: at BTN1+2, there should be an RTS, not a .BYT $2c. The code as published will change the desired .Z=0 to .Z=1 if location $01A9 is zero, and will change the desired .N=1 to .N=0 if bit 7 of $01A9 is 0."

# TeleColumn

## Transactor Online Conference
## Sunday, April 26, 8:00 PM EST
## on CompuServe.
## Subject: Writing Articles that GET PUBLISHED!

We get dozens of articles every month, many we can't use. Why? The reasons come up as fast as the articles come in. But it doesn't have to be that way. Much of this is covered in our Transactor Writer's Guide, which was four years in the making (available free - send SASE). However, since printing the guide we've acquired enough new criteria for '86'ing an article to make the next guide double the size of the first, and we're quite sure we'll find more.

At this conference we'll discuss everything from topic ideas to preparing the package. Typesetting is one of most major tasks, and many programs we receive are good, but too difficult to typeset. We'll discuss how you can adjust programs you've already written to reduce our preparation efforts and increase your chances for seeing them in print. And if they make it into print, YOU get paid. . . and isn't *that* what it's all about? Darn right.

Never written an article before? Many articles we print are the author's first attempt. Writing is often easier than talking, because you have time to think about what you'll say. So if you've been considering a writing hobby for some extra cash, join us in this conference. Even if you're not planning to write computer related material, we'll help you put together a submission that will be irresistible at any publication.

The conference will be held in CBMCOM, The Commodore Communications Forum. Once on CompuServe, type "GO CBMCOM" and enter "CO" at the main function prompt. It starts at 8:00 PM EST on Sunday, April 26, and myself, Rich, Chris and Nick will see you there!

### Transactor Programs on CompuServe

We're gradually getting all of the Transactor Disks uploaded to CompuServe. You can find them in Data Library 17 of CBMPRG, The Commodore Programming Forum. Data Library 16 of CBMPRG will contain all of the programs that have been published in the Bits section, and those few that have appeared in the Letters column.

Since filenames on CompuServe are limited to six characters (the extension will always be ".BIN" ie. uploaded with Xmodem), they won't always be very descriptive. So unless you know the exact name we chose to upload the file, it might get difficult locating a specific program. Therefore, we carefully designed a format of keywords that makes them easy to find. Here's an example:

V7 I04 P70 JAN87 C64 TRACE44.SRC HI RES TRACER PAL SOURCE

The keywords start with the Volume, Issue, and Page numbers, followed by the date on the cover. The Page number is the page that the listing starts on, NOT the first page of the article.

Next is the machine(s) the program will run on. Some programs will run on any machine, in which case this will be "BASIC". The exact name of the file as it appears on the Transactor Disk comes next ("TRACE44.SRC"), and we suggest you use this name once it's downloaded to make it more identifiable when talking with others who have the program, but obtained it elsewhere. At the moment we're not sure how to show where the "exact filename" ends, so this will be duplicated in the file description that is displayed below the keywords. Following the filename are some extra keywords that will describe the general purpose of the file if the filename leaves a little too much to the imagination.

Now when you're reading a Transactor, and you just can't wait to get a copy of a program, sign on to CompuServe, "GO CBMPRG", and enter DL17 at the main function prompt (DL16 for Bits). If "Mini-Tracer" is the program you want, as in the

above example, enter the following at the DL prompt:

bro /key:v7 i04 p70

. . .and the file should appear instantly. The above would show you the source file. If you only want the Loader, you would enter:

bro /key:v7 i04 p**69**

Occasionally there may be two or three programs that start on the same page. If the one you want doesn't show up first, just keep hitting Return at the prompts and eventually it will.

HOWEVER, please note that although the above is an accurate example, Mini-Tracer had not been uploaded as of this writing.

### The Magazine Display Area

After starting work on the Display Area that we've been mentioning over the last four issues, we soon realized that if we were going to refer to programs from within the articles on the service, we should know what the name is before we refer to it. We could have chose names prior to uploading the programs, but then the articles would refer to files in the DLs that aren't yet there. So we decided that the programs should be safely in the DLs before we refer to them by name. When all the Transactor programs are uploaded, work will resume on the Display Area, which should be sometime in April.

## CAD30.BIN

**by Steve Nye [70366,1316]**

**a drawing/drafting program for the Commodore 64**

Did you ever wish you could try out that new floorplan before moving all the furniture? Or print a map to your house for those out–of–town guests? Need to create a bar or pie chart for that sales meeting tomorrow? All this and more is available for the price of downloading from CompuServe's CBMART SIG.

CAD is a drawing/drafting program for the Commodore 64. Originally designed to do electronic schematics, it has evolved into one of the most complete drawing packages available. The ease with which it is modified and expanded has allowed the users of the package to be major contributors to its development.

Aside from the usual drawing functions (line, box, circle, erase, etc.), there were from the outset three main requirements for the package, an easily expanded and modified command set, three hires screens resident in RAM, and the ability to access a library of pre–designed figures.

The ease with which BASIC is modified made it the obvious choice for a programming language, but the lack of high resolution commands had to be overcome. A package of USR calls from the old Commodore SIG, USRML, made it possible to access machine language routines for hires work while keeping the main structure of the program in BASIC.

CAD is actually a package of eighty–three short programs. The command function programs are linked together by a command processor, and only one is loaded at a time. The use of linked programs keeps the amount of memory needed by BASIC to a minimum and allows a virtually unlimited command set. Since the CAD programs only need to pass numeric variables, the linking process is a simple one; it is only necessary to ensure that the first program loaded is the longest in the package.

The pre–drawn figure access was a simple programming problem. The figures are stored on disk as X and Y displacements from a reference point. When the figure is recalled, these displacements are added to the present position of the cursor and the pixel at that location is set. The size of the figure is limited to plus or minus 127 pixels and a total of 255 'on' pixels can be included in each figure. The number of figures on a disk is limited only by the number of available directory entries. Thanks to user support there are now over thirty figure libraries, including furniture for floorplans, USGS map symbols, electronic schematic symbols, plumbing fixtures, and a number of font sets for lettering hires screens.

A package that has not received as much attention as CAD, but is closely related, is GRAPH. This package allows drawing high resolution pie charts, bar charts, and line plots. Using a dynamic keyboard technique it even allows graphing of a user–input equation. Since GRAPH initially boots the entire CAD support library, it is possible to move from GRAPH to CAD by simply changing the disk. This allows the user to modify, letter, and otherwise manipulate the graph.

The CAD and GRAPH packages are available from CBMART Data Library 12 under the filenames CAD30.BIN and GRAPHV-.BIN. There are numerous support files, such as the figure sets, and a number of CADGames designed to be played using CAD. In fact, the support files for CAD became so numerous that now Data Library 12 is devoted exclusively to CAD. Support for the package is available through regularly scheduled conferences and the message section of the SIG.

## Another Online Odyssey

**by Marte Brengle, Glendale, CA**

**Check out The Source and expand your online world.**

Ranjan Bose's article in the May '87 Transactor gave a good overview of three of the most popular national online services for Commodore owners. One other large national service whose name doesn't usually come up when one thinks of topics of interest to Commodore owners is The Source.

In the past, The Source's target "audience" has been business–oriented and mostly IBM–using, rather than the kinds of people who use the "consumer" services provided by CompuServe or QuantumLink. That is changing these days, and a large factor influencing that change is the new Commodore–oriented Special Interest Group (SIG). Any of you who used to visit The Transactor on Viewtron may already be familiar with the folks who run the Commodore SIG on The Source. It's the Indepen-

dent Computer User Group (ICUG), the same crew who ran Speakeasy, Inner Works, and For Starters on Viewtron.

The Source provides the same kinds of services as the other major networks – information, public domain software, games, and the like – and has just added a real–time conferencing area, called Chat. Those of you who have used CompuServe's "CB Simulator" or any of the "CO" areas in the Forums may find that Chat is even easier to use.

CompuServe's interactive conferences are an exercise in mental gymnastics, unless you are using a terminal program with a split screen, since your messages can get totally tangled up with other people's on the screen. One of the first things CIS CB users learn to do is press ctrl–V (repeatedly!) to straighten out their messages so they can see what they've typed so far. No need to do that in Source Chat. Since The Source is accessed primarily through packet–switching networks like Telenet and Tymnet, a few simple keystrokes will take you back to "network level," switch you to half duplex, return to Chat and show you your own messages without their being overrun by everyone else's. And with Source's command structure, you can even set yourself up a "personal command key" (as was possible on Viewtron) to do all that "preparation" automatically as it takes you into Chat. (More about that in a moment.) It's like getting the best of QuantumLink or PlayNET (each message is an entity unto itself and you don't have to unscramble them) with the best of the ASCII networks (no proprietary software required). And nicer still, you don't pay extra for using higher baud rates in Source Chat. It's billed at the 300 baud rate no matter how fast you choose to go (up to 2400 baud at present).

The Source also features one of the most sophisticated "message–board" systems around, called Participate(tm). You may be surprised at the kinds of things that are possible in Participate, or Parti as it's called. Much like CIS, messages follow each other in "threads", but on CIS, only the Sysops can move messages from one topic to another. Not so in Parti. Messages can be branched off, moved around, sent to other topics, used to begin new topics, or whatever your imagination desires – no special Sysop powers required. There are private topics on both CIS and Parti, but on Parti anyone can start one, and can invite anyone of his or her own choosing.

One of the nicest things about the late, lamented Viewtron was that it allowed you to set up your own personal "command keys" to take you anywhere in the system. If you didn't feel like typing ICUGSIG SPEAKEASY every time, for example, you could program the combination =S to take you there. The Source lets you do the same thing. You can program as many commands into a sequence as you need, to get you anywhere on the system with just one keystroke, if you choose. Here's an example of one of my own custom keys. I've programmed the letter C to switch me to half duplex, take me into ICUGSIG Chat (the special Chat area run by the Commodore SIG), check to see who else is there, give me an "alias" (or "handle"), make sure I can see my own messages, and enter the ICUG Reception channel. The listing looks really complex when you see it on the screen, but it works like a charm:

```
C term –half;icug chat.1015 –s –al marte –di me –j 31
```

More complete instructions for setting up personal command keys are found in the Source manual, and also in Parti under the topic "Custom Keys."

You can also set up a file to run automatically each time you sign onto The Source, if there are certain things you'd like to do every time you sign on. For example, my file turns my "custom keys" on, and then checks to see what's in my mailbox. But that's not all you can do. Ever get tired of seeing the system's "what next" prompts? You can reprogram even that to suit your mood. Instead of the standard "arrow" that The Source uses for a Command Level prompt, I have it set up to say "What now?" And instead of the question–mark–and–arrow combination that means you made a mistake somewhere, my prompt says "Uh oh!" My sign–on file sets that up as well. You can find instructions for all that in the Source manual under "Customizing The Command Level Prompt," or in Parti in the "Custom Keys" topic.

Although I've mentioned the Source manual several times here, and it is for the most part a good investment, the fastest way to find help for what you need to do on line is generally to ask someone. The Parti section in the manual, for example, is out of date (as is the Parti section in Charlie Bowen and Dave Peyton's excellent book "How to Get The Most Out Of The Source"). And ICUGSIG is a great place to find all kinds of help with using the system. There are quite a few "help with Parti" topics set up under ICUGSIG Exchange in Parti, and the SIG also provides a "New User Survival Kit" under the main SIG menu that's a gold mine of information. It covers such things as navigation commands, "how to download/upload," how to use SourceMail, how to use the Members' Directory, and a complete and thorough tutorial on how to use Parti. And there's a new "Ask The Experts" section that's much like the one on Viewtron, where Commodore users can get information on hardware, software, or any other computer–related problem.

You can upload to and download from the software libraries with a variety of protocols, including two variations on "capture buffer," Xmodem, Kermit, and, for Amiga users, Sliding Window Kermit in ATerm (which is also available for downloading from the ICUG library).

ICUGSIG on The Source provides software, advice, and information for all Commodore computer users. There's a large library of C64, C128, and Amiga software, and a large group of sysops who can provide help with almost anything imaginable. You can recognize the Sysops online by their ID numbers, which all have a SIG prefix. The ICUG Sysops' ID numbers range from SIG015 to SIG025.

As Ranjan Bose's article pointed out, there is a wealth of information and services available online for Commodore owners. But it's not all on CompuServe and QuantumLink! Check out The Source and expand your online world.

# A Real Shuffle Subroutine

**Thomas W. Gurley**
**Wills Point, Texas**

How many times have you ever played Blackjack where the dealer threw all the cards up in the air and then picked them up in random order and dealt them out? Not very many, I'm sure. But that is just about the way most computer programs shuffle cards. The shuffle subroutine of the program uses a pseudo-random number generator to rearrange the order of the cards each time they are dealt. On the other hand, a human dealer would not shuffle like that. He would divide the deck into two packets and riffle one half with the other. Most dealers will let one to five cards fall from each packet until all the cards are riffled. He will then split the pack again and repeat the shuffle several times. Each new deal thus depends (in a complicated way) upon all previous hands.

The card shuffle subroutine presented here more nearly approximates the actions of a human dealer.

This subroutine starts with a "fresh deck" by assigning the order to the cards from 1 to 52 when first run. It then shuffles several times just as a real dealer would do. Before each new deal, the cards will be shuffled a random number of times from 3 to 13. You can change this feature (and others) to suit yourself.

I wanted to analyze the game of Blackjack to find out why "runs" seem to occur. In real-life Blackjack, the cards will sometimes hit in runs and eventually they will "patternize" so that one of the players (or the dealer) will always hit. It may require hundreds of deals before a "run" will occur but it will happen in total defiance of all logic. This subroutine will do exactly the same thing!

All you need do to obtain more realistic hands is substitute this subroutine for the one currently being used in your card game. You will have to adjust the array names to match your program, but that should be easy.

The Blackjack program I modified had two calls to a shuffle subroutine. It randomly filled an array with numbers from 1 to 52 each time. It then went on to assign card values to the numbers in the arra. All that was necessary to implement the new shuffle subroutine was to put a GOSUB 8000 as the first line of the original randomizer and then jump past it (on return) to the "card value" routine.

It works really well and the hands seem to appear more like those in real Blackjack games I've been in. I hope you can use this subroutine as well.

```
JK  8000 rem **shuffle subroutine**
PA  8010 rem       8/27/86
AE  8020 rem **dimension arrays**
EO  8030 dim c%(52),c1(26),c2(26):rem (0) not used
GI  8040 print" it f1 to start"
MM  8050 if peek(197)<>4 then 8050: rem time is part
         of random factor
AH  8060 rem set up the deck
MN  8070 forx=1to52:c%(x)=x:next:formm=1to10
         :gosub8130:next
ND  8080 x=rnd(-ti):rem enter here for all subsequent
         shuffles
IL  8090 sh=int(10*rnd(x)+1)+2: rem change the
         '10' to suit yourself
BF  8100 formm=1tosh:gosub8130:d=0: rem shuffle
         random times
CL  8110 next
ED  8120 end: replace this with 'return'
EC  8130 rem **a real shuffle**
EH  8140 c=0:ca=0:cb=0:cc=0:cd=0
FP  8150 print" . . .shuffle. . ..shuffle. . ."
NI  8160 fori=1to26:c1(i)=c%(i):c2(i)=c%(i+26):next
         :rem split deck into two
FE  8170 gosub8400:gosub8250:ifcb>=26then8360
NF  8171 rem deal 1st card from right packet so 1st
         card changes (just like real!)
EC  8180 gosub8400:gosub8200:ifca>=26then8300
DB  8190 goto8170
JB  8200 rem **riffle cards from left half**
AD  8210 forc=1tor:ca=ca+1:ifca>26thenc=7:return
DM  8220 cd=cd+1
FI  8230 c%(cd)=c1(ca)
KB  8240 next:return
MD  8250 rem**riffle cards from right half**
EG  8260 forc=1tor:cb=cb+1:ifcb>26thenc=5:return
FP  8270 cd=cd+1
AH  8280 c%(cd)=c2(cb):next
OH  8290 return
LH  8300 rem **shuffle balance of left**
MM  8310 forc=cbto26:cb=cb+1:ifcb>26thenreturn
HC  8320 cd=cd+1
CK  8330 c%(cd)=c2(cb):next
AL  8340 return
GD  8350 rem **shuffle balance of right**
EP  8360 forc=cato26:ca=ca+1:ifca>26thenreturn
JF  8370 cd=cd+1
BN  8380 c%(cd)=c1(ca):next
CO  8390 return
JE  8400 x=rnd(-ti)
OG  8410 r=4*rnd(x)+1
AA  8420 return
ML  9000 rem ***this is for you to verify
         there is one and only one of each
JN  9010 rem number in the array  c%(x).
         use 'goto 9000' and note that
HE  9020 rem each left hand number occurs
         once only.
JI  9030 forx=1to56:forjj=1to52:ifc%(jj)=xthen
         printx;jj
EE  9040 next:next:end
```

# Function Manipulation: Roots and Integrals

Eric Giguere
Waterloo, Ontario

One of the computer's strongest points is its mathematical prowess. In terms of speed alone, computers have revolutionized modern mathematics. Things which used to be drudgery for human mathematicians are handled without complaint by a properly programmed computer. The following article deals with two useful applications of computers: estimation of function roots and integrals.

## Roots

A function is a way of expressing the fact that one quantity depends upon another. By definition, the dependent quantity must be unique. We usually see a function expressed in the form:

$$f(x) = x + 1$$

For a certain value of x, out pops another value, in this case x+1. If the function is to be plotted, we may see it expressed in the following form:

$$y = x + 1$$

Thus given a point x, we can find a related point y and plot the corresponding coordinate on a graph such as the one in Figure 1a.

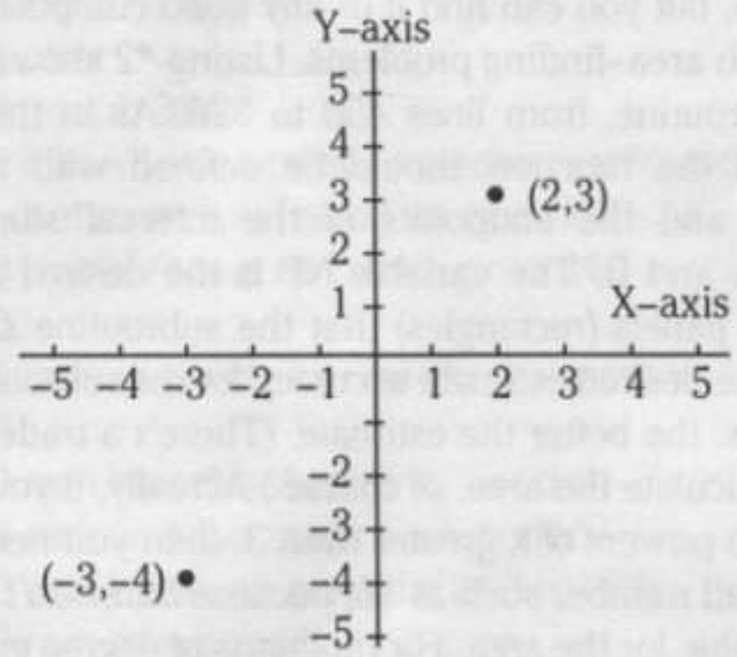



**Figure 1a: The Catesian Coordinate System**

A root or zero of a function is defined as the point where the function is zero, or where the graph of the function crosses the x–axis. A function may have one root, several roots, or no roots. The function $y=x+1$ has a root at the point $x=-1$, since $y=(-1)+1=0$. But a function such as $y=x^2+1$ has no root, since y will never be zero no matter what value of x we choose.

There are times when it is useful to know where the roots of a function are, if it has any. Some functions are particularly easy in this respect, but others are a lot harder. This is where the computer comes in. Several methods have been perfected over the years to yield accurate root estimates for a given function. The one I will be presenting you with is called the bisection method.

## BISECT

The idea behind the bisection method is very simple. Take a function f(x) that has a root somewhere in the interval [A,B] (that is, it has a root at a point x such that A <= X <= B), as in Figure 1b. We then divide this interval into two equal subintervals and throw away the subinterval that doesn't contain the root (see Figure 1c). We then repeat the process on the remaining subinterval, and continue on in this fashion until we are within a certain distance (or tolerance) of the root. This gives us the approximation for the root.

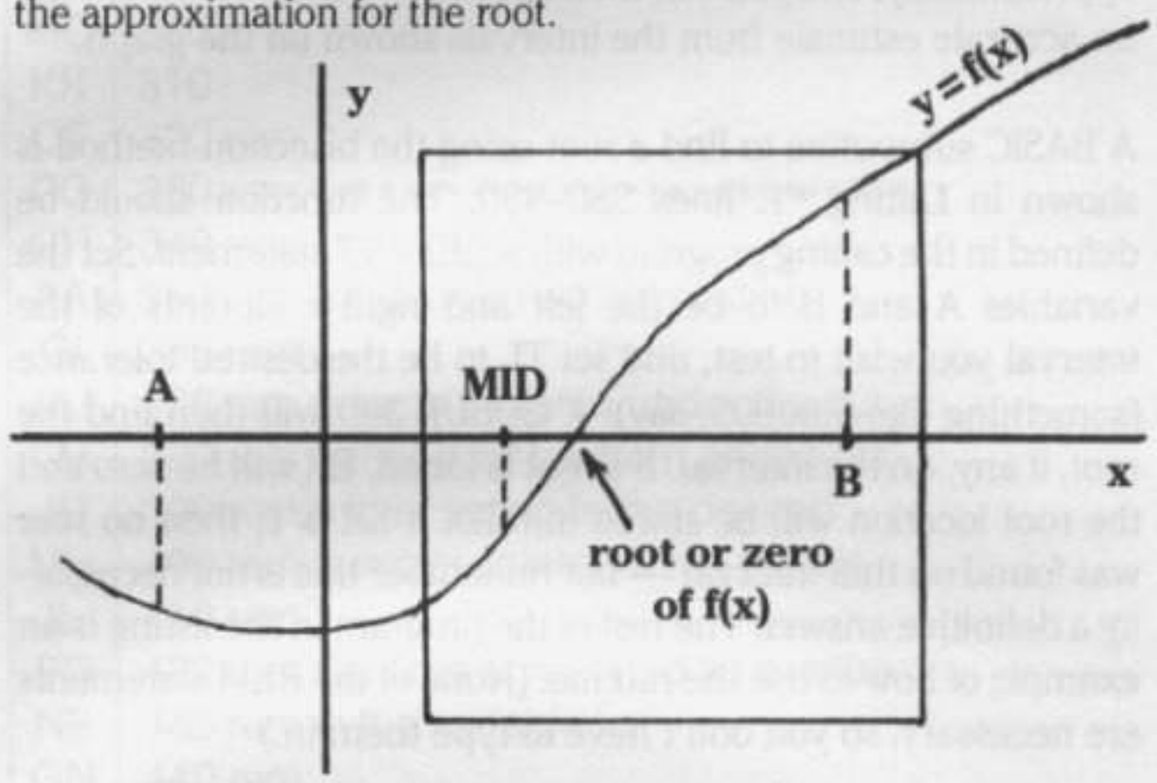



**Figure 1b): BISECT algorithm**

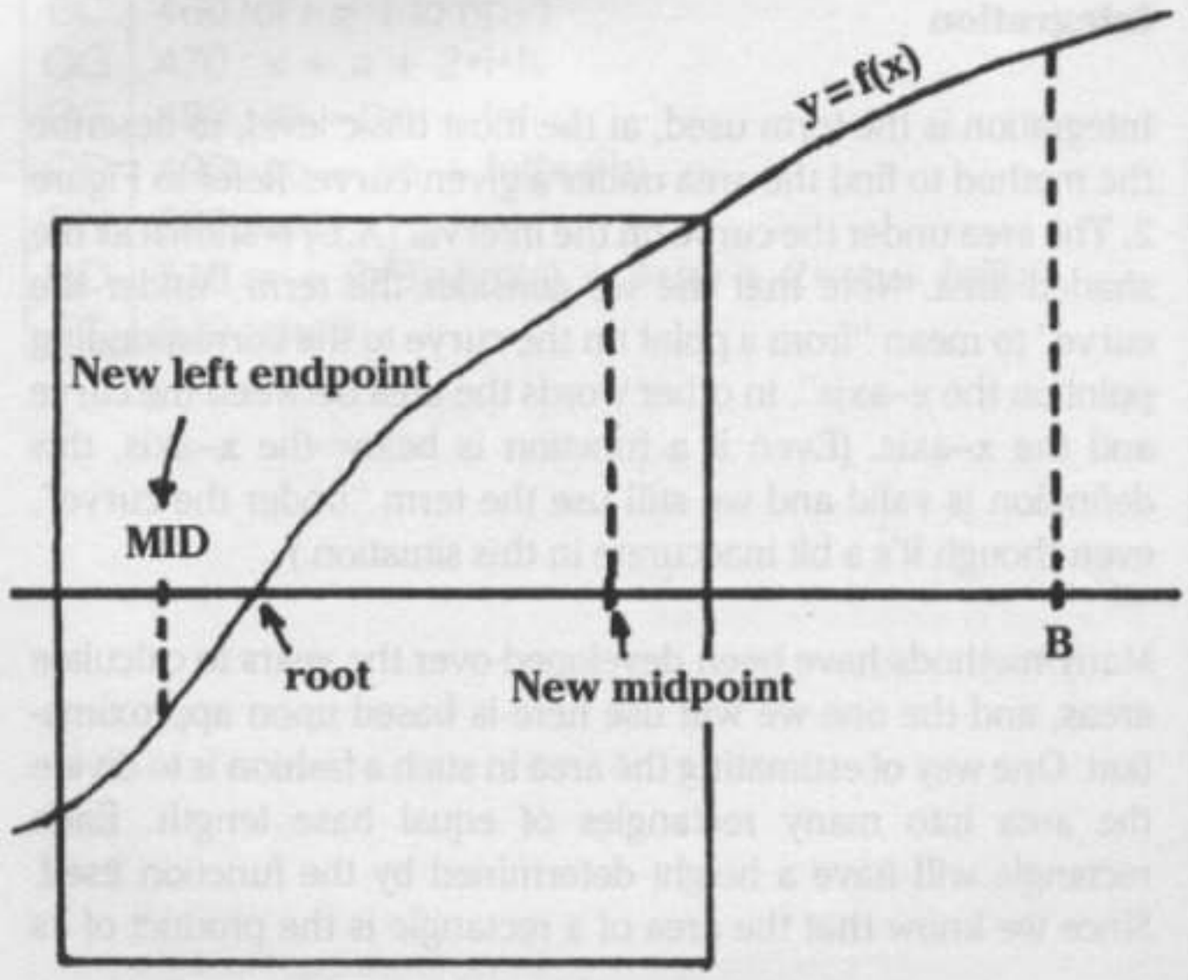



**Figure 1c): Blow-up of 1b)**

We need to know one thing before going ahead and writing the algorithm for this method: how do we know if an interval contains a zero or not? The easiest way is to compare the function values at the two endpoints of a specific interval. If there is a root in the interval, then one y–value will lie below the x–axis and the other will be above it. Thus if we are looking at the interval [A,B], there will be a root if the following condition holds:

$$f(A)*f(B) < 0$$

(Remember that if you multiply a negative number by a positive one you get a negative answer, and a positive one otherwise.) This is the test we will use in the algorithm.

NOTE: The above test is not accurate for all intervals of a function. If, for example, a function crosses the x–axis twice on the interval [A,B], then $f(A)*f(B) > 0$, even though there are two roots in the interval. Thus you must be careful of which interval you choose to get an accurate example. Plotting the graph over a large interval can often give you a good idea where the roots are approximately, and you can then use the bisection method to get an accurate estimate from the intervals shown on the graph.

A BASIC subroutine to find a root using the bisection method is shown in Listing #1, lines 380–430. The function should be defined in the calling program with a DEFFNF statement. Set the variables A and B to be the left and right endpoints of the interval you wish to test, and set TL to be the desired tolerance (something like 0.00005, say). A GOSUB 380 will then find the root, if any, on the interval. If a root is found, ER will be zero and the root location will be stored in MID. If ER is 1, then no root was found on that interval –– but remember this is not necessarily a definitive answer. The rest of the program in the listing is an example of how to use the routine. (None of the REM statements are necessary, so you don't have to type them in.)

## Integration

Integration is the term used, at the most basic level, to describe the method to find the area under a given curve. Refer to Figure 2. The area under the curve on the interval [A,B] is shown as the shaded area. Note that the we consider the term "under the curve" to mean "from a point on the curve to the corresponding point on the x–axis", in other words the area between the curve and the x–axis. (Even if a function is below the x–axis, this definition is valid and we still use the term "under the curve", even though it's a bit inaccurate in this situation.)

Many methods have been developed over the years to calculate areas, and the one we will use here is based upon approximation. One way of estimating the area in such a fashion is to divide the area into many rectangles of equal base length. Each rectangle will have a height determined by the function itself. Since we know that the area of a rectangle is the product of its base and height, we simply sum the individual rectangle areas to get an approximation for the entire area under the curve. The approximation gets better as the number of rectangles increases (and hence the base length decreases).

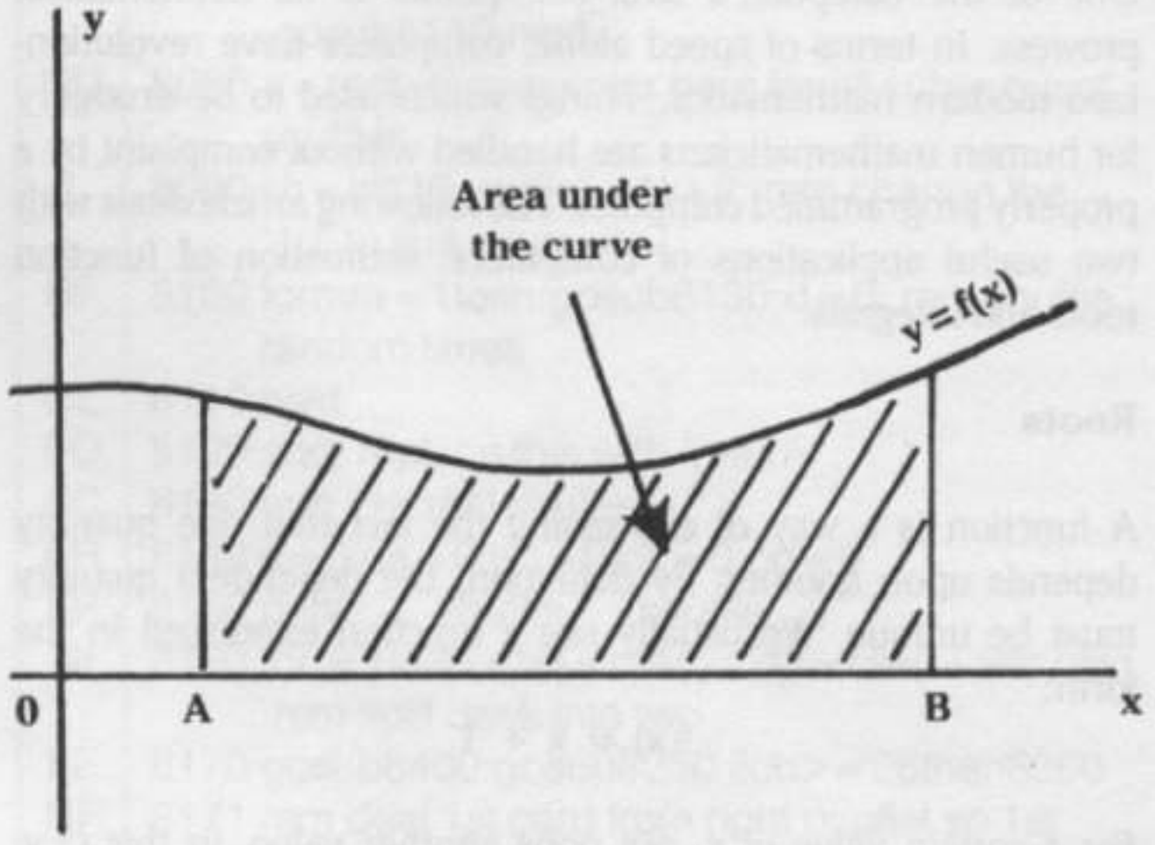



**Figure 2: Area under the curve between points A and B**

The actual area–finding routine described in this article is a variation of the approximation method detailed above called Simpson's Rule. I won't go into the theory as to how the rule works here, but you can find it in any good computer textbook dealing with area–finding problems. Listing #2 shows the actual BASIC subroutine, from lines 450 to 520. As in the bisection subroutine, the function should be defined with a DEF FN statement, and the endpoints of the interval stored in the variables A and B. The variable NP is the desired number of integration panels (rectangles) that the subroutine should use. This sets the desired estimate accuracy for the subroutine –– the larger NP is, the better the estimate. (There's a tradeoff in time taken to calculate the area, of course.) Actually, if your function FNF has no powers of x greater than 3, then you need only set NP to a small number such as 16, because Simpson's Rule gives an exact value for the area. For functions of degree greater than 3 (ie, they have powers of x that are greater than three), Simpson's Rule gives an estimate, an estimate which improves with larger values of NP. As before, the rest of Listing #2 gives an example of how to use the subroutine.

## Final Notes

Computer algorithms such as the above have made themselves very useful for solving many physical problems. Examples abound that require a person to know the roots or integrals of a function, and sometimes the most efficient way to find these is to use a computer. While the answers such algorithms may give are

limited both by the computer's accuracy (usually found as round-off errors) and the inherent inaccuracy of the algorithms, they are usually good enough for the task (especially since the time required to find an accurate answer using manual methods may outweigh the benefits of having such an accurate answer). If you're interested in other such approximation techniques, many computer science textbooks are good sources of such information and usually the algorithms they give may be converted to BASIC without much trouble. I know that I've found the integration routine particularly useful in checking my answers in my university calculus courses. Maybe you can find a good use for them, too.

**Listing 1: BISECT**

```
CD  100 rem sample program to find the
FL  110 rem roots of the function
GJ  120 rem
IG  130 rem y=(x-3)(x-1)(x+4)
KK  140 rem
NE  150 rem which we know are x=1,3, and -4.
OL  160 rem
IC  170 rem try the intervals (0,2), (2,0)
PH  180 rem and (-5,-3)
CD  190 :
LB  200 def fnf(x) = (x-3)*(x-1)*(x+4)
EA  210 print
FI  220 input "left endpoint";a
DA  230 input "right endpoint";b
MC  240 input "tolerance";tl
MC  250 print
HM  260 gosub 380
DN  270 if er=1 then print "no zero was found
        between";a;"and";b: goto 210
NA  280 print "root is at:";mid: goto 210
GJ  290 :
IE  300 rem ****************************
KK  310 :
KF  320 rem bisect subroutine, used to
EB  330 rem find the zero of a function
BC  340 rem between points a and b, with
AB  350 rem tolerance tl. if no zero
JB  360 rem exists, er=1.
AJ  370 rem
CC  380 fl=fnf(a): fr=fnf(b): er=0
PB  390 if fl*fr>0 then er=1: return
FO  400 mid = (a+b)/2
EK  410 if abs(a-mid)<= tl then return
IA  420 fm = fnf(mid)
PM  430 if fl*fm<= 0 then b = mid: goto 400
JK  440 a = mid: fl = fm: goto 400
```

**Listing 2: Approximate Area using Simpson's Rule**

```
LE  100 rem program example using simpson's
NE  110 rem rule to find the area under
EK  120 rem the curve
AK  130 rem
OI  140 rem   y = 1 + x + x↑2
EL  150 rem
AO  160 rem from x=0 to x=2. by direct
MO  170 rem integration the answer we
DI  180 rem should get will be 20/3, or
MD  190 rem approx. 6.66667
GO  200 rem
IJ  210 rem first define the function
AF  220 :
BC  230 deffnf(x)= 1 + x + x*x
EG  240 :
CC  250 rem now call the subroutine to
MB  260 rem find the area
CI  270 :
HE  280 a = 0: b = 2: np = 16: gosub 450
HO  290 print "the area from";a;"to";b;"is:";ar
MC  300 end
KK  310 :
OF  320 rem
CG  330 rem ******************************
CH  340 rem
AA  350 rem simpson's rule subroutine
GI  360 rem
MJ  370 rem returns the area under the curve
LM  380 rem defined by fnf, from endpoints
JD  390 rem a to b. set np to the desired
MH  400 rem number of integration panels
IL  410 rem
PD  420 rem the area is returned as a value
NE  430 rem in the variable 'ar'
GN  440 rem
FE  450 h=(b-a)/(2*np): se=0: so=fnf(a+h)
BC  460 for i = 1 to np-1
OG  470 : x = a + 2*i*h
CG  480 : se = se + fnf(x)
OD  490 : so = so + fnf(x+h)
DN  500 next i
GD  510 ar = (h/3)*(fnf(a) + 4*so + 2*se + fnf(b))
EC  520 return
```

# Full Array Math Operations On The 64

Richard Richmond
Springfield, Ohio

*. . .When manipulating large arrays or carrying out repetitive operations, these routines can significantly reduce the execution time of a program.*

This article describes a group of math functions that perform the basic math operations like "add, subtract, multiply and divide" on an entire array with just one SYS command. Using these routines, code like the following;

```
100 for i=0 to 100
110 a(i) = a(i)*c
120 next i
```

can be reduced to SYS SA+18,C,A(0), where A is the name of an array and C is a separate variable. Not only do these routines provide simpler code, but they are faster than the corresponding operation in BASIC statements. These routines are written in machine language and will carry out the operations from ten to fifty times faster than BASIC. When manipulating large arrays or carrying out repetitive operations, these routines can significantly reduce the execution time of a program. Because this program makes extensive use of the floating point routines in ROM, it also provides examples of how to use those routines in your own programs.

## The New Functions

Table 1 lists the SYS commands for these new utilities. Also listed are the math functions performed by each of the functions. You can see from the list that the program can perform the four math functions +, –, *, and / on either two arrays or one array and a separate variable. Some of the routines are provided to take into account the non–reciprocity of certain operations. For example, "A=A+B" is the same, mathematically, as "A=B+A". But, "A=A–C" is not the same as "A=C–A". Table 1 shows that separate routines are provided for the following non–reciprocal operations:

| | |
|---|---|
| A=A–V | A=A–B |
| A=V–A | A=B–A |
| A=A/V | A=A/B |
| A=V/A | A=B/A |

Throughout this article and in the examples, the labels A and B will be used to indicate arrays and the label V to indicate a single variable. Note that when a variable is indicated that variable could also be one of the elements of an array.

## The Rules and Exceptions

Because most of these routines follow the same conventions and format, we will first discuss those and later point out the exceptions.

Two arguments, separated by a comma, are passed to the utility. The second argument will be one of the arrays that will be operated on and the array in which the results will be stored. These routines are written to operate on floating point arrays and variables.

When working on an array, two addresses are used by the routines. The address for the end of the array in the second argument is found by the program and used to determine when the operation is completed. The other address is used to tell the procedure where in the second array to start the operation. In Table 1, the array elements are x (such as a(x)). Normally, x would be =0. In that case, the procedure begins with the first, or (0), element of the array and operates on every element in the array. A non–zero element would cause the routine to skip some of the elements. For example, SYS SA,V,A(0) would cause every element in A to be set equal to V. But, SYS SA,V,A(2) would leave A(0) and A(1) unchanged and all of the rest of the elements would be equal to V.

With this type of flexibility, some precaution is necessary. For the two array operations, the number of elements in the first argument from the element specified to the end of array must be, at least, equal to the number of elements in the second argument from the specified element to the end of that array. The routines only look for the ending address of the second array. If the utility runs out of elements in the first array before reaching the end of the second array, the program continues but the results from that point on will not be valid. No error or system crash should result, just useless data.

Arrays must be dimensioned and the elements should be defined before the utilities are called. The utility can be used to define elements if they are all the same value. Variables do not have to be predefined. But if they are not, they will be set to zero the first time that they are used in a SYS to one of the routines. This program depends heavily upon the floating point routines in ROM. Therefore, the rules for math operations in BASIC still apply and the routine will return error messages such as "DIVISION BY ZERO" if illegal operations are attempted.

As mentioned earlier, there are exceptions to the above guidelines. Two of the routines are not limited to floating point numbers. "EQUB" works on all three types of variables (integer, string and floating point) and sets one array equal to a second. "INSERT" is the second utility that also works with all variable types. This routine puts the value of the first argument into the array at the location specified, A(x). All of the following elements are shifted down one. That is, A(x+1)=A(x),A(x+2)=A(x+1) and so on. The last element in the array is lost.

Also, notice that "SQUARE" has only one argument. The elements of the array are squared and stored back into the array, so no second argument is needed.

For most of these routines, the value of the first argument is not changed and should be defined before being used in one of the

routines, unless that value is to be zero. The exceptions are 'mn' and 'mx' used in "MIN" and "MAX". The minimum (mn) or maximum (mx) value of the array will be returned and the initial value of the variable does not matter.

### A Look At The Program

The source code for these routines is listed in Program 3. Rather than discuss it line-by-line, we will look at some general characteristics and some of the subroutines or modules used by almost all of the routines. The labels at the beginning of the listing refer to the ROM routines that do the bulk of the floating point math. Next in the listing is the "JUMP" table. This technique makes it much easier to insert or delete code or to make changes in the routines because the "SYS" statement in BASIC does not change. Also, the "SYS" address for new routines that may be added later can be easily determined.

To obtain as much speed as possible, zero page addressing is used extensively. But, there is not enough zero page free memory for our purposes. Therefore, "SZPAGE" saves the contents of a block of zero page memory to a safe location and "RESTOREZ" loads the values back to zero page before returning to BASIC.

The "TEST" routines are used to step through the array five bytes at a time (5 bytes are used to store a floating point number) and to set a flag when the end of the array is reached. "MOD1" looks for the first argument in the variable or array storage area and creates it if not found. The address of this variable is then saved. Next, "MOD2" finds the address of the array element in the second argument and the address of the end of this array. These values are also stored for use by the routines. In general, the steps followed when one of the routines is called are:

1. Save zero page values
2. Use "MOD" routine to find the arguments
3. Addresses are loaded to the registers
4. Numbers are loaded into the Floating Point Accumulator (FAC)
5. The required math is performed between the FAC and memory
6. Results in FAC are stored in memory
7. Steps 3–6 are repeated for entire array
8. Zero page is restored and control returned to BASIC

### Using The Program

First, type in and save Program 2. When this program is run, a ML program file is written to disk with the name "ARRAY.FNC". Load this program with:

```
LOAD"ARRAY.FNC",8,1
```

Because these utilities were conceived to provide time savings for graphics programs, they are located to be compatible with graphics utility programs that load at $C000 (49152). The starting address (SA) for these utilities is 51800. Once these utilities are loaded, type "NEW" and enter or load your BASIC program. Near the beginning of your program, define SA=51800. SA can then be used for the SYS statements listed in Table 1.

For examples of how these routines work, look at Program 1. Not every routine is illustrated in Program 1, but if you run the program and examine the output, you will get a good idea of what these routines do. Note that in the next to last example array B is squared, but the element (x) in the SYS command is 4. Compare that print out with the previous one and you will see that the first 4 elements (x=0 to 3) are not changed and the remaining elements (x=4 to 10) have been squared. The examples in Program 1 should give you a good starting point for your own experiments in how to use these routines.

Some of the routines, such as "MIN" and "MAX" are especially useful when graphing. Using these routines, the maximum and minimum values of an array could be used to "scale" the data before it is displayed.

These utilities are fast. They are reasonably "goof proof" yet still allow a good deal of flexibility. Any program that manipulates large arrays should benefit from the increased speed possible with these utilities. The use of jump tables and a modular approach to programming the different operations make it fairly simple to add other functions to the program. Obvious ideas for additional routines would be a square root function and one to raise the array to a given power. Because both EQUB and INSERT work on all three types of variables, you might want to write another program that only has these two routines in it.

Although I first thought of these routines as aids for graphics programs, they would be equally useful in a database or spreadsheet type of program. For that type of use, another possible function would return the sum of an array. I would be interested in hearing from anyone who has suggestions for improvements or additions to this program.

**TABLE 1: Summary of Commands**

| Format | Routine | Operation |
|---|---|---|
| SYS SA,V,A(X) | EQUV | A=V |
| SYS SA+3,B(Y),A(X) | EQUB | A=B |
| SYS SA+6,V,A(X) | PLUV | A=A+V |
| SYS SA+9,B(Y),A(X) | PLUB | A=A+B |
| SYS SA+12,V,A(X) | SUBV | A=A–V |
| SYS SA+15,B(Y),A(X) | SUBB | A=A–B |
| SYS SA+18,V,A(X) | MULV | A=A•V |
| SYS SA+21,B(Y),A(X) | MULB | A=A•B |
| SYS SA+24,V,A(X) | DIVV | A=A/V |
| SYS SA+27,B(Y),A(X) | DIVB | A=A/B |
| SYS SA+30,V,A(X) | SBUV | A=V–A |
| SYS SA+33,B(Y),A(X) | SBUB | A=B–A |
| SYS SA+36,V,A(X) | DVIV | A=V/A |
| SYS SA+39,B(Y),A(X) | DVIB | A=B/A |
| SYS SA+42,MX,A(X) | MAX | MAX(A) |
| SYS SA+45,MN,A(X) | MIN | MIN(A) |
| SYS SA+48,A(X) | SQUARE | A=A↑2 |
| SYS SA+51,V,A(X) | INSERT | A(X)=V |

NOTE: SA =starting address (51800)
IF X or Y>0 in SYS command, routine will skip over elements in array to (X). Elements with (a)<X will be unmodified.

**Program 1: Example**

```
AK  10 sa=51800: dim a(10),b(10)
IE  20 print: for i=0 to 10: a(i)=i: next
EN  30 print: print "set b=a": sys sa+3,a(0),b(0)
CM  40 print: for i=0 to 10: print b(i);: next
II  50 v1=2: sys sa+6,v1,b(0): print:
       print "set b=b+2"
```

```
ED  60 for i=0 to 10: printb(i);: next
MC  70 v2=25: print: print " insert 25 at a(5) ":
        sys sa+51,v2,a(5)
FE  80 for i=0 to 10: printa(i);: next
CP  90 print: print "set b=b*a ": sys sa+21,a(0),b(0)
MF  100 for i=0 to 10: print b(i);: next
OA  110 print: print "square b - (b=b↑2)"
FP  130 print "starting with element x=4":
        sys sa+48,b(4)
EI  140 for i=0 to 10: print b(i);: next
NM  150 print: print "using an undefined variable":
        sys sa,v3,a(0)
JD  160 print: for i=0 to 10: printa(i);: next
```

**Program 2: Creates disk file "ARRAY.FNC"**

```
IK  100 rem m/l loader for array math
BE  110 for j=51800 to 52778
JH  120 read x : ch=ch+x : next
ED  130 if ch<> 141610 then print "checksum
        error" : end
IP  140 print "data ok. . . creating 'array.fnc'"
OH  150 hi=int(51800/256):lo=51800-hi*256
HB  160 open 1,8,1,"0:array.fnc"
PO  170 print#1,chr$(lo)chr$(hi);
MM  180 restore
OI  190 for i=51800 to 52778
NN  200 read x : print#1,chr$(x); : next
FK  210 close1
KP  220 rem
DA  230 data  76, 233, 202,  76, 169, 204,  76,   5
PC  240 data 203,  76, 215, 203,  76,  40, 203,  76
BJ  250 data 250, 203,  76, 110, 203,  76,  29, 204
GE  260 data  76, 145, 203,  76,  64, 204,  76,  75
EE  270 data 203,  76,  99, 204,  76, 180, 203,  76
PJ  280 data 134, 204,  76,  37, 205,  76, 107, 205
GG  290 data  76, 177, 205,  76, 212, 205, 255, 113
GE  300 data 255, 255, 255, 255,   0,   0,  88, 202
JL  310 data   1,   8,  64,   0,   0,   3,   0,   3
HG  320 data  64, 160,  12, 185, 191,   0, 153, 148
MN  330 data 202, 136,  16, 247,  96, 160,  12, 185
DN  340 data 148, 202, 153, 191,   0, 136,  16, 247
PN  350 data  96, 166, 181, 164, 182,  32, 212, 187
AN  360 data  96, 165, 183,  24, 105,   5, 133, 183
II  370 data 165, 184, 105,   0, 133, 184, 165, 185
BP  380 data  24, 105,   5, 133, 185, 165, 186, 105
DJ  390 data   0, 133, 186, 197, 252, 208,   8, 165
CM  400 data 185, 197, 251, 208,   2,  24,  96,  56
BK  410 data  96,  32, 161, 202,  32,  20, 205, 165
LB  420 data 183, 164, 184,  32, 162, 187, 166, 185
OO  430 data 164, 186,  32, 212, 187,  32, 206, 202
BP  440 data 176, 244,  76, 173, 202,  32, 161, 202
CM  450 data  32,  20, 205, 165, 185, 164, 186, 133
FC  460 data 181, 132, 182,  32, 162, 187, 165, 183
DB  470 data 164, 184,  32, 103, 184,  32, 185, 202
HB  480 data  32, 206, 202, 176, 230,  76, 173, 202
LE  490 data  32, 161, 202,  32,  20, 205, 165, 183
MF  500 data 164, 184,  32, 162, 187, 165, 185, 133
IF  510 data 181, 164, 186, 132, 182,  32,  80, 184
OA  520 data  32, 185, 202,  32, 206, 202, 176, 230
PB  530 data  76, 173, 202,  32, 161, 202,  32,  20
MG  540 data 205, 165, 185, 133, 181, 164, 186, 132
AK  550 data 182,  32, 162, 187, 165, 183, 164, 184
HC  560 data  32,  80, 184,  32, 185, 202,  32, 206
DH  570 data 202, 176, 230,  76, 173, 202,  32, 161
MC  580 data 202,  32,  20, 205, 165, 185, 133, 181
BK  590 data 164, 186, 132, 182,  32, 162, 187, 165
LP  600 data 183, 164, 184,  32,  40, 186,  32, 185
BK  610 data 202,  32, 206, 202, 176, 230,  76, 173
HM  620 data 202,  32, 161, 202,  32,  20, 205, 165
JO  630 data 183, 164, 184,  32, 162, 187, 165, 185
IN  640 data 133, 181, 164, 186, 132, 182,  32,  15
KL  650 data 187,  32, 185, 202,  32, 206, 202, 176
PA  660 data 230,  76, 173, 202,  32, 161, 202,  32
HP  670 data  20, 205, 165, 185, 133, 181, 164, 186
FA  680 data 132, 182,  32, 162, 187, 165, 183, 164
OM  690 data 184,  32,  15, 187,  32, 185, 202,  32
JP  700 data 206, 202, 176, 230,  76, 173, 202,  32
BM  710 data 161, 202,  32,  20, 205, 165, 183, 164
LD  720 data 184,  32, 162, 187, 165, 185, 133, 181
HD  730 data 164, 186, 132, 182,  32, 103, 184,  32
IE  740 data 185, 202,  32, 193, 202, 176, 230,  76
LF  750 data 173, 202,  32, 161, 202,  32,  20, 205
JF  760 data 165, 183, 164, 184,  32, 162, 187, 165
NC  770 data 185, 133, 181, 164, 186, 132, 182,  32
OH  780 data  80, 184,  32, 185, 202,  32, 193, 202
HE  790 data 176, 230,  76, 173, 202,  32, 161, 202
LH  800 data  32,  20, 205, 165, 183, 164, 184,  32
FK  810 data 162, 187, 165, 185, 133, 181, 164, 186
JK  820 data 132, 182,  32,  40, 186,  32, 185, 202
CI  830 data  32, 193, 202, 176, 230,  76, 173, 202
JK  840 data  32, 161, 202,  32,  20, 205, 165, 183
KL  850 data 164, 184,  32, 162, 187, 165, 185, 133
AM  860 data 181, 164, 186, 132, 182,  32,  15, 187
EH  870 data  32, 185, 202,  32, 193, 202, 176, 230
NH  880 data  76, 173, 202,  32, 161, 202,  32,  20
KM  890 data 205, 165, 185, 164, 186, 133, 181, 132
OP  900 data 182,  32, 162, 187, 165, 183, 164, 184
DK  910 data  32,  80, 184,  32, 185, 202,  32, 193
BN  920 data 202, 176, 230,  76, 173, 202,  32, 161
KI  930 data 202,  32,  20, 205, 165, 185, 133, 181
PP  940 data 164, 186, 132, 182,  32, 162, 187, 165
FG  950 data 183, 164, 184,  32,  15, 187,  32, 185
MA  960 data 202,  32, 193, 202, 176, 230,  76, 173
LB  970 data 202,  32, 161, 202,  32,  20, 205, 160
MM  980 data   0, 177, 183, 145, 185, 230, 183, 208
NL  990 data   2, 230, 184, 230, 185, 208,   2, 230
AD 1000 data 186, 165, 186,  32, 219, 202, 176, 231
II 1010 data  76, 173, 202,  32, 253, 174,  32, 158
OG 1020 data 173, 165,  71, 133, 185, 165,  72, 133
DI 1030 data 186, 165,  47, 133, 251, 165,  48, 133
MJ 1040 data 252, 160,   0, 177, 251, 197,  69, 208
HG 1050 data  11, 200, 177, 251, 197,  70, 208,   5
OI 1060 data  32, 251, 204,  96, 200,  32, 251, 204
LG 1070 data  76, 225, 204, 200, 177, 251, 133, 253
GD 1080 data 200, 177, 251,  24, 101, 252, 133, 252
GF 1090 data 165, 251,  24, 101, 253, 133, 251, 144
CC 1100 data   2, 230, 252,  96,  32, 253, 174,  32
DN 1110 data 139, 176, 165,  71, 133, 183, 165,  72
II 1120 data 133, 184,  76, 203, 204,  32, 161, 202
BN 1130 data  32,  20, 205, 165, 185, 164, 186,  32
CH 1140 data 162, 187, 162, 142, 160, 202,  32, 212
DO 1150 data 187,  32, 206, 202, 165, 185, 164, 186
CM 1160 data  32, 162, 187, 162, 142, 160, 202, 138
```

```
BL 1170 data  32,  91, 188,  48,   7, 162, 142, 160
FL 1180 data 202,  32, 212, 187,  32, 206, 202, 176
GJ 1190 data 227, 162, 142, 160, 202, 138,  32, 162
DA 1200 data 187, 166, 183, 164, 184,  32, 212, 187
HM 1210 data  76, 173, 202,  32, 161, 202,  32,  20
IA 1220 data 205, 165, 185, 164, 186,  32, 162, 187
PA 1230 data 162, 142, 160, 202,  32, 212, 187,  32
LO 1240 data 206, 202, 165, 185, 164, 186,  32, 162
AC 1250 data 187, 162, 142, 160, 202, 138,  32,  91
MP 1260 data 188,  16,   7, 162, 142, 160, 202,  32
IC 1270 data 212, 187,  32, 206, 202, 176, 227, 162
IC 1280 data 142, 160, 202, 138,  32, 162, 187, 166
GI 1290 data 183, 164, 184,  32, 212, 187,  76, 173
EA 1300 data 202,  32, 161, 202,  32, 203, 204, 165
JE 1310 data 185, 164, 186, 133, 181, 132, 182,  32
KJ 1320 data 162, 187, 165, 185, 164, 186,  32,  40
BG 1330 data 186,  32, 185, 202,  32, 206, 202, 176
HL 1340 data 230,  76, 173, 202,  32, 161, 202,  32
MH 1350 data  20, 205, 169,   5, 133, 191, 165,  70
AI 1360 data  10, 144,  11, 169,   2, 133, 191, 165
MK 1370 data  69,  10, 176,   2, 230, 191, 165, 251
OL 1380 data  56, 229, 191, 133, 251, 165, 252, 233
FL 1390 data   0, 133, 252, 165, 251,  56, 229, 191
OI 1400 data 133, 253, 165, 252, 233,   0, 133, 254
HN 1410 data 164, 191, 136, 177, 253, 145, 251, 136
OB 1420 data  16, 249, 165, 252, 197, 186, 208, 214
IP 1430 data 165, 251, 197, 185, 208, 208, 164, 191
KN 1440 data 136, 177, 183, 145, 251, 136,  16, 249
EB 1450 data  76, 173, 202
```

## Program 3: PAL Source Code

```
AK 100 rem array math functions
IA 110 rem c richard richmond
KC 120 rem 308 rosewood ave.
KC 130 rem springfield, ohio
BF 140 rem 45506
EL 150 rem
GD 160 open8,8,1,"0:array.obj"
LH 170 sys700
HN 180 .opt o8
AA 190 *=$ca58
AH 200 memfac  =  $bba2      ;memory to fac1
JM 210 facmem  =  $bbd4      ;fac1 to memory
PK 220 compar  =  $bc5b      ;compare memory to fac1
LH 230 memplu  =  $b867      ;add memory to fac1
ML 240 memmul  =  $ba28      ;mult fac1 by memory
BB 250 memsub  =  $b850      ;sub fac1 from memory
PN 260 memdiv  =  $bb0f      ;divide fac1 by memory
JG 270           jmp eqv     ; a()=v 'starting address
II 280           jmp eqb     ; a()=b() 'sa+3
CP 290           jmp plv     ; a()=a()+v 'sa+6
JG 300           jmp plb     ; a()=a()+b() 'sa+9
FI 310           jmp sbv     ; a()=a()-v 'sa+12
FI 320           jmp sbb     ; a()=a()-b() 'sa+15
NL 330           jmp mlv     ; a()=a()*v 'sa+18
LK 340           jmp mlb     ; a()=a()*b() 'sa+21
GN 350           jmp dvv     ; a()=a()/v 'sa+24
EN 360           jmp dvb     ; a()=a()/b() 'sa+27
KP 370           jmp bsv     ; a()=v-a() 'sa+30
JN 380           jmp bsb     ; a()=b()-b() 'sa+33
KB 390           jmp vdv     ; a()=v/a() 'sa+36
AP 400           jmp vdb     ; a()=b()/a() 'sa+39
NP 410           jmp max     ; v=max(a()) 'sa+42
LB 420           jmp min     ; v=min(a()) 'sa+45
OK 430           jmp square  ; a()=a()↑2 'sa+48
CN 440           jmp insert  ; insert v at a() 'sa+51
KO 450 dummy     *=*+6
AH 460 zpage     *=*+13
LG 470 szpage                ; routine to save
HM 480           ldy #$0c    ; zero page memory
CG 490 sz1
BC 500           lda $00bf,y
FC 510           sta zpage,y
DM 520           dey
AF 530           bpl sz1
IA 540           rts
KP 550 reset                 ;routine to reset
FO 560           ldy #$0c    ;aero page memory
AG 570 restorez
NC 580           lda zpage,y
JL 590           sta $00bf,y
DB 600           dey
GE 610           bpl restorez
NJ 620           rts         ;exit
LO 630 store                 ;store fac1
KB 640           ldx $b5     ;to memory
LD 650           ldy $b6     ; specified at
ML 660           jsr facmem  ; $b5,$b6
KI 670           rts
MK 680 test1                 ;this portion
JB 690           lda $b7     ;increments the
BP 700           clc         ;second array
OM 710           adc #$05    ;pointers by
EO 720           sta $b7     ;5
CJ 730           lda $b8
JP 740           adc #$00
EO 750           sta $b8
EF 760 test                  ;routine to
AH 770           lda $b9     ;increment
BF 780           clc         ;first array
OB 790           adc #$05    ;pointers by
KD 800           sta $b9     ;5
NP 810           lda $ba
JE 820           adc #$00
PE 830           sta $ba
JC 840 test2                 ;and check
JP 850           cmp $fc     ;for the end
LC 860           bne cont    ;of array
BC 870           lda $b9
DJ 880           cmp $fb
GJ 890           bne cont
KP 900           clc
KH 910           rts
JB 920 cont                  ;if not to end
OC 930           sec         ;set carry bit
IJ 940           rts
CM 950 eqv                   ;a()=v
BJ 960           jsr szpage  ;store zero page
DE 970           jsr mod1    ;get addresses
CH 980           lda $b7     ;address o
ID 990           ldy $b8     ;v
AH 1000          jsr memfac  ;load v to fac1
IP 1010 eqv1
CE 1020          ldx $b9     ;address of
BD 1030          ldy $ba     ;a()
JH 1040          jsr facmem  ;fac1 to a(x)
IG 1050          jsr test    ;check if done
OI 1060          bcs eqv1    ;no continue
BO 1070          jmp reset   ;yes exit routine
DJ 1080 plv                  ;a()=a()+v
DO 1090          jsr szpage
MJ 1100          jsr mod1
EG 1110 plv2
JB 1120          lda $b9     ;load address
IG 1130          ldy $ba     ;of next a()
CM 1140          sta $b5     ;pointer for
CF 1150          sty $b6     ;store routine
GG 1160          jsr memfac  ;1st element to fac1
GH 1170          lda $b7     ;address of
GP 1180          ldy $b8     ;v
HN 1190          jsr memplu  ;add v to fac1
LJ 1200          jsr store   ;results to a()
GL 1210          jsr test
PA 1220          bcs plv2
GM 1230          jmp reset
MB 1240 sbv                  ;a()=a()-v
DI 1250          jsr szpage
MD 1260          jsr mod1
LO 1270 sbv1
DL 1280          lda $b7     ;load address
LO 1290          ldy $b8     ;of v
PL 1300          jsr memfac  ;v to fac1
HN 1310          lda $b9     ;load address
MN 1320          sta $b5     ;of
NF 1330          ldy $ba     ;a()
MI 1340          sty $b6
PH 1350          jsr memsub  ;a()added to fac1
JE 1360          jsr store   ;result to a()
GF 1370          jsr test
KJ 1380          bcs sbv1
GG 1390          jmp reset
DA 1400 bsv                  ;a()=v-a()
DC 1410          jsr szpage
MN 1420          jsr mod1
```

```
MJ | 1430 bsv1
LF | 1440        lda  $b9
HJ | 1450        sta  $b5
HO | 1460        ldy  $ba
OA | 1470        sty  $b6
BD | 1480        jsr  memfac    ;fac1 = a( )
ED | 1490        lda  $b7       ;address
NL | 1500        ldy  $b8       ;of v
BM | 1510        jsr  memsub    ;fac1 = v-a( )
BI | 1520        jsr  store     ;a( ) = fac1
GP | 1530        jsr  test
LE | 1540        bcs  bsv1
GA | 1550        jmp  reset
PG | 1560 mlv                   ;a( ) = a( )•v
DM | 1570        jsr  szpage
MH | 1580        jsr  mod1
ND | 1590 mlv1
LP | 1600        lda  $b9
EO | 1610        sta  $b5       ;address
KN | 1620        ldy  $ba       ;of a( )
OK | 1630        sty  $b6
BN | 1640        jsr  memfac    ;fac1 = a( )
EN | 1650        lda  $b7       ;address
NF | 1660        ldy  $b8       ;of v
HD | 1670        jsr  memmul    ;fac1 = a( )•v
BC | 1680        jsr  store     ;a( ) = fac1
GJ | 1690        jsr  test
IO | 1700        bcs  mlv1
GK | 1710        jmp  reset
MB | 1720 dvv                   ;a( ) = a( )/v
DG | 1730        jsr  szpage
MB | 1740        jsr  mod1
JO | 1750 dvv1
NE | 1760        lda  $b7       ;adress
LM | 1770        ldy  $b8       ;of v
HK | 1780        jsr  memfac    ;fac1 = v
GG | 1790        lda  $b9       ;address
IE | 1800        sta  $b5       ;of a( )
FE | 1810        ldy  $ba
MG | 1820        sty  $b6
PL | 1830        jsr  memdiv    ;fac1 = a( )/v
BM | 1840        jsr  store     ;a( ) = fac1
GD | 1850        jsr  test
DJ | 1860        bcs  dvv1
GE | 1870        jmp  reset
EO | 1880 vdv                   ;a( ) = v/a( )
DA | 1890        jsr  szpage
ML | 1900        jsr  mod1
HH | 1910 vdv1
IO | 1920        lda  $b9       ;address
KM | 1930        sta  $b5       ;of a( )
HM | 1940        ldy  $ba
OO | 1950        sty  $b6
BB | 1960        jsr  memfac    ;fac1 = a( )
EB | 1970        lda  $b7       ;address
NJ | 1980        ldy  $b8       ;of v
EH | 1990        jsr  memdiv    ;fac1 = v/fac1
BG | 2000        jsr  store     ;a( ) = fac1
GN | 2010        jsr  test
BC | 2020        bcs  vdv1
GO | 2030        jmp  reset
JP | 2040 plb                   ;a( ) = a( )•b( )
DK | 2050        jsr  szpage
MF | 2060        jsr  mod1
DN | 2070 plb1
CI | 2080        lda  $b7       ;address
IJ | 2090        ldy  $b8       ;of b( )
OJ | 2100        jsr  memfac    ;fac1 = b( )
JP | 2110        lda  $b9
CO | 2120        sta  $b5       ;address
IN | 2130        ldy  $ba       ;of a( )
MK | 2140        sty  $b6
JL | 2150        jsr  memplu    ;fac1 = fac1•a( )
BA | 2160        jsr  store     ;a( ) = fac1
HE | 2170        jsr  test1     ;increment b pointer then a
PI | 2180        bcs  plb1
GI | 2190        jmp  reset
AI | 2200 sbb                   ;a( ) = a( )-b( )
DE | 2210        jsr  szpage
MP | 2220        jsr  mod1
LF | 2230 sbb1
CC | 2240        lda  $b7       ;address
ID | 2250        ldy  $b8       ;of b( )
OD | 2260        jsr  memfac    ;fac1 = b( )
JJ | 2270        lda  $b9
CI | 2280        sta  $b5       ;address
IH | 2290        ldy  $ba       ;of a( )
ME | 2300        sty  $b6
GL | 2310        jsr  memsub    ;fac1 = a( )-fac1
HH | 2320        jsr  store     ;a( ) = 1
AN | 2330        jsr  test1     ;increment b then a
OB | 2340        bcs  sbb1
GC | 2350        jmp  reset
DD | 2360 mlb                   ;a( ) = a( )•b( )
DO | 2370        jsr  szpage
MJ | 2380        jsr  mod1
NA | 2390 mlb1
CM | 2400        lda  $b7       ;address
IN | 2410        ldy  $b8       ;of b( )
ON | 2420        jsr  memfac    ;fac1 = b( )
JD | 2430        lda  $b9
CC | 2440        sta  $b5       ;address
IB | 2450        ldy  $ba       ;of a( )
MO | 2460        sty  $b6
EO | 2470        jsr  memmul    ;fac1 = fac1•a( )
BE | 2480        jsr  store     ;a( ) = fac1
ND | 2490        jsr  test1     ;increment pointers
MM | 2500        bcs  mlb1
GM | 2510        jmp  reset
AO | 2520 dvb                   ;a( ) = a( )/b( )
DI | 2530        jsr  szpage
MD | 2540        jsr  mod1
JL | 2550 dvb1
CG | 2560        lda  $b7       ;address
IH | 2570        ldy  $b8       ;of b( )
OH | 2580        jsr  memfac    ;fac1 = b( )
JN | 2590        lda  $b9
CM | 2600        sta  $b5       ;address
IL | 2610        ldy  $ba       ;of a( )
MI | 2620        sty  $b6
IG | 2630        jsr  memdiv    ;fac1 = fac1•a( )
BO | 2640        jsr  store     ;a( ) = fac1
NN | 2650        jsr  test1     ;increment pointers
HH | 2660        bcs  dvb1
GG | 2670        jmp  reset
BH | 2680 bsb                   ;a( ) = b( )-a( )
DC | 2690        jsr  szpage
MN | 2700        jsr  mod1
ME | 2710 bsb1
LF | 2720        lda  $b9
KI | 2730        ldy  $ba       ;address
EP | 2740        sta  $b5       ;of a( )
OA | 2750        sty  $b6
BD | 2760        jsr  memfac    ;fac1 = a( )
ED | 2770        lda  $b7       ;address
KE | 2780        ldy  $b8       ;of b( )
HJ | 2790        jsr  memsub    ;fac1 = b( )-fac1
BI | 2800        jsr  store     ;a( ) = fac1
NH | 2810        jsr  test1     ;increment pointers
PA | 2820        bcs  bsb1
GA | 2830        jmp  reset
OA | 2840 vdb                   ;a( ) = b( )/a( )
DM | 2850        jsr  szpage
MH | 2860        jsr  mod1
HO | 2870 vdb1
LP | 2880        lda  $b9
EO | 2890        sta  $b5       ;address
KN | 2900        ldy  $ba       ;of a( )
OK | 2910        sty  $b6
BN | 2920        jsr  memfac    ;fac1 = a( )
EN | 2930        lda  $b7       ;address
KO | 2940        ldy  $b8       ;of b( )
PB | 2950        jsr  memdiv    ;fac1 = b( )/fac1
BC | 2960        jsr  store     ;a( ) = fac1
HM | 2970        jsr  test1
FK | 2980        bcs  vdb1
GK | 2990        jmp  reset
IG | 3000 eqb                   ;a( ) = b( )
DG | 3010        jsr  szpage
MB | 3020        jsr  mod1
MI | 3030 eqb1
DG | 3040        ldy  #$00
NI | 3050        lda  ($b7),y   ;1st byte of b
CE | 3060        sta  ($b9),y   ;into a
GI | 3070        inc  $b7       ;increment
JL | 3080        bne  eqb2      ;address
HH | 3090        inc  $b8       ;of b
DN | 3100 eqb2
EL | 3110        inc  $b9       ;increment
FO | 3120        bne  eqb3      ;address
HL | 3130        inc  $ba       ;of a
MP | 3140 eqb3
FD | 3150        lda  $ba       ;hi byte of a
BL | 3160        jsr  test2     ;end of array
AJ | 3170        bcs  eqb1      ;no continue
PB | 3180        jmp  reset     ;yes exit routine
OB | 3190 mod2                  ;find address of a
JF | 3200        jsr  $aefd     ;skip comma
HP | 3210        jsr  $ad9e     ; routine
AB | 3220        lda  $47       ;lo byte of
```

```
DE 3230          sta  $b9        ;a address
HB 3240          lda  $48        ;hi byte of
PG 3250          sta  $ba        ;a address
ED 3260          lda  $2f        ;start of
CO 3270          sta  $fb
OO 3280          lda  $30        ;array storage
JP 3290          sta  $fc
LC 3300 again                    ;search array storage
BH 3310          ldy  #$00
GD 3320          lda  ($fb),y
PB 3330          cmp $45         ;for name
OM 3340          bne  step2      ;of
GP 3350          iny
OF 3360          lda  ($fb),y
AD 3370          cmp $46         ;'a' array
BH 3380          bne  step       ;routine returns
BI 3390          jsr  step1      ;with ending address
PA 3400          rts             ;of a( )in $fb,$fc
KD 3410 step2
MD 3420          iny
IL 3430 step
AF 3440          jsr  step1
JD 3450          jmp  again
IN 3460 step1                    ;routine to
KF 3470          iny             ;skip through
JP 3480          lda  ($fb),y    ;array memory
GN 3490          sta  $fd        ;from one
OG 3500          iny             ;array to next
EP 3510          lda  ($fb),y
GD 3520          clc
NJ 3530          adc  $fc
DP 3540          sta  $fc
ML 3550          lda  $fb
OF 3560          clc
IM 3570          adc  $fd
IB 3580          sta  $fb
AP 3590          bcc  st2
BB 3600          inc  $fc
EI 3610 st2
AB 3620          rts
GJ 3630 mod1                     ;routine to find b( )
BB 3640          jsr  $aefd      ;skip comma
OG 3650          jsr  $b08b      ; routine
AJ 3660          lda  $47        ;to find v,b
NC 3670          sta  $b7        ;or create
HJ 3680          lda  $48        ;variable
CP 3690          sta  $b8        ;if not found
OK 3700          jmp  mod2
MG 3710 max                      ;v = maximum of a( )
JC 3720          jsr  szpage
CO 3730          jsr  mod1
HF 3740          lda  $b9
JN 3750          ldy  $ba
CN 3760          jsr  memfac     ;fac1 = a(0)
PI 3770          ldx  #<dummy    ;store in
DC 3780          ldy  #>dummy
EE 3790          jsr  facmem     ;'dummy'
EN 3800          jsr  test
BN 3810 max2
EJ 3820          lda  $b9        ;next address
MH 3830          ldy  $ba        ;of a( )
JG 3840          jsr  memfac     ;fac1 = a( )
JG 3850          ldx  #<dummy
DH 3860          ldy  #>dummy
CN 3870          txa
AG 3880          jsr  compar     ;compare
IJ 3890          bmi  max3       ;a( ) with 'dummy'
FK 3900          ldx  #<dummy    ;fac1 larger
FK 3910          ldy  #>dummy
JI 3920          jsr  facmem     ;then 'dummy' = fac1
KE 3930 max3
KO 3940          jsr  test       ;done
KM 3950          bcs  max2       ;no continue
PJ 3960          ldx  #<dummy    ;yes
BO 3970          ldy  #>dummy
BF 3980          txa             ;fac1 = 'dummy'
IA 3990          jsr  memfac
OF 4000          ldx  $b7        ;address
LI 4010          ldy  $b8        ;of v
EH 4020          jsr  facmem     ;v = fac1
GL 4030          jmp  reset
KJ 4040 min                      ;v = minimum of a( )
DH 4050          jsr  szpage
MC 4060          jsr  mod1
OE 4070          lda  $b9        ;address
PM 4080          ldy  $ba        ;of a(0)
AI 4090          jsr  memfac     ;store
HG 4100          ldx  #<dummy    ;a(0) into
NG 4110          ldy  #>dummy
OI 4120          jsr  facmem     ;'dummy'
OB 4130          jsr  test
LA 4140 min2
AC 4150          lda  $b9        ;address of
PH 4160          ldy  $ba        ;next a( )
HI 4170          jsr  memfac     ;load into fac1
OE 4180          ldx  #<dummy    ;and
NL 4190          ldy  #>dummy
MB 4200          txa
NJ 4210          jsr  compar     ;compare with 'dummy'
BK 4220          bpl  min3       ;fac1<'dummy
EH 4230          ldx  #<dummy    ;then 'dummy'
PO 4240          ldy  #>dummy
BO 4250          jsr  facmem     ; = fac1
EI 4260 min3
KK 4270          jsr  test
BE 4280          bcs  min2       ;'dummy = min(a)
BC 4290          ldx  #<dummy
LC 4300          ldy  #>dummy
KI 4310          txa
DC 4320          jsr  memfac     ;transfer
ND 4330          ldx  $b7        ;'dummy'
MJ 4340          ldy  $b8        ;to
AI 4350          jsr  facmem     ;v
AA 4360          jmp  reset
IP 4370 square                   ;a = a*a
NL 4380          jsr  szpage
KH 4390          jsr  mod2
OE 4400 squ1
CK 4410          lda  $b9        ;address
KM 4420          ldy  $ba        ;of a( )
LD 4430          sta  $b5
IK 4440          sty  $b6
BG 4450          jsr  memfac     ;a( ) to fac1
HC 4460          lda  $b9
JK 4470          ldy  $ba
LH 4480          jsr  memmul     ;fac1 = a*a
EA 4490          jsr  store
LH 4500          jsr  test       ;increment pointer
PO 4510          bcs  squ1
AK 4520          jmp  reset
GO 4530 insert                   ;a(x) = v
MB 4540          jsr  szpage     ;following
HD 4550          jsr  mod1       ;elements
JO 4560          lda  #$05       ;moved down
DJ 4570          sta  $bf        ;a(max) = a(max-1)
FB 4580          lda  $46        ;continue
BF 4590          asl  a          ;until
EB 4600          bcc  ins1       ;a(x+1) = a(x)
BD 4610          lda  #$02       ;then a(x) = v
MC 4620          sta  $bf        ;routine will
NK 4630 ins1                     ;automatically
LJ 4640          lda  $fb        ;use proper offset
CK 4650          clc             ;for variable type
LP 4660          sbc  $bf        ;works with
EN 4670          sta  $fb        ;strings (a$)
IC 4680          lda  $fc        ;integers (a%)
AC 4690          sbc  #$00       ;or
PC 4700          sta  $fc        ;floating point
EE 4710          lda  $fb
GO 4720          clc
MG 4730          sbc  $bf
GK 4740          sta  $fd
PG 4750          lda  $fc
LM 4760          sbc  #$00
HM 4770          sta  $fe
OO 4780          ldy  $bf
BH 4790          dey
KL 4800 ins2
AB 4810          lda  ($fd),y
AF 4820          sta  ($fb),y
JJ 4830          dey
CD 4840          bpl  ins2
JN 4850          lda  $fe
EB 4860          cmp $ba
KC 4870          bne  ins1
EP 4880          lda  $fd
KB 4890          cmp $b9
IE 4900          bne  ins1
AH 4910          ldy  $bf
DP 4920          dey
ND 4930 ins3
CF 4940          lda  ($b7),y
CN 4950          sta  ($fb),y
LB 4960          dey
IL 4970          bpl  ins3
MG 4980          jmp  reset
KF 4990 .end
```

# Faster Square Root For The Commodore 64

**E.J. Schmahl**
**Bowie, Maryland**

*. . .not only is it fast, it is much more accurate!*

If you time the C–64 functions built into your C–64 BASIC, you will find that the SQR function is very much slower than +,–,*, or /. In fact it slower than EXP or LOG because it uses both of those functions, which are rather slow themselves. However, you don't have to be resigned to slow square roots, because it turns out that there are well–known algorithms that run faster than the SQR built into your C–64 ROM.

To make this possible, I have written a short machine language square root routine which takes the first variable in the BASIC variable area of memory and replaces it with its square root. The program uses the BASIC ROM floating point functions which move, add and divide numbers. (No LOGs or EXPs are used.)

This routine can be run and compared with the standard SQR function in ROM:

```
10 input a
20 print sqr(a);:sys49152
30 print a: goto 10
```

You will find that the new square root is about 2.5 times faster than the SQR in ROM, quite an improvement! But not only is it fast, it is much more accurate. Just try printing out the square roots of these squares: 26569, 21316689, 84309124, and 0.779689. The new square root gives the exact answers (163, 4617, 9182, and 0.883), but the ROM SQR routine does not! The reason is that the EXP function is not very accurate (6 to 8 digits instead of 9) for certain numbers. (This inaccuracy problem has been cured in the C–128.)

Note that, to keep things simple, (and fast) the new SQR operates only on the very first variable in BASIC. Thus whatever variable appears first in the program is changed into its square root. Make sure you declare your desired variable first with an equality (e.g. 1 x=1) or a DIMension statement (e.g. 1 dim x).

The BASIC loader loads the machine language into the area starting at 49152. You can change the variable AD in line 60 to 828 or 9*4096 or whatever you like. Line 70 of the loader program will modify the non–relocatable machine language command "lda tabl,x" appropriately for the new starting address.

For those interested in the algorithm used for computing the square root, it is Newton's method, with a good first approximation from a 16–entry table. In BASIC, the equivalent program would be:

```
10 input a
20 if a=0 then 90
30 if a<0 then 120
40 input "first approx";approx
50 x=approx
60 x=(x+a/x)/2
70 k=k+1
80 if k<4 then 60
90 a=x
100 print "sqr=" a
110 k=0: goto 10
120 print "illegal quantity error"
```

Enthusiasts of greater speed in the C–64 should note that there is room for improvement in most of the other BASIC functions. The favored method for approximating functions in the C–64 ROM seems to be power series. As the present example shows, there are sometimes better ways to generate functions. A challenge to you programmers out there: try some new methods and speed up your programs!

### BASIC Loader

```
MC 100 rem  a=64.1:sys49152:print a
CP 110 rem  where a is the first variable
HI 120 rem  result of sqr will be in a
GE 130 ad=49152:ch=0
PJ 140 for i=0 to 122:read x
GI 150 poke ad+i,x:ch=ch+x:next
CE 160 if ch<>13691 then print "error in data
       statements":end
NL 170 hi=int((ad+107)/256)
CI 180 poke ad+59,hi:poke ad+58,ad+107-hi*256
JB 190 rem new table address for relocated p
JB 200 data 160,  3, 177,  45,  48,  98, 136, 177
AP 210 data  45, 240,  92, 165,  45,  24, 105,   2
PG 220 data 133, 170, 165,  46, 105,   0, 133, 171
```

```
GN 230 data 160,   0, 132,  93, 132,  94, 132,  95
LN 240 data 132,  96, 177, 170, 106, 176,   4, 162
MH 250 data 128, 134,  93, 105,  64, 133,  92, 200
DN 260 data 177, 170,   5,  93,  74,  74,  74,  74
IF 270 data 170, 189, 107,  32, 133,  93, 169,   4
DG 280 data 133, 172, 169,  92, 160,   0,  32, 162
MI 290 data 187, 165, 170, 164, 171,  32,  15, 187
DI 300 data 169,  92, 160,   0,  32, 103, 184, 198
KI 310 data  97,  32, 199, 187, 198, 172, 208, 233
DM 320 data 166, 170, 164, 171,  32, 212, 187,  96
KI 330 data  76,  72, 178,   3,  11,  18,  25,  32
FP 340 data  38,  44,  50,  58,  69,  79,  89,  98
IL 350 data 107, 115, 123
```

**PAL Source Code**

```
FD 100 rem faster square root pal source
PN 110 open8,8,1,"0:fast sqr.obj
JE 120 sys 700
FK 130 .opt o8
NH 140 * = $c000
BP 150 ; faster square root
BH 160 ; takes first basic variable and
LP 170 ; returns square root in its place
IG 180 ; uses newton's method with a good
KG 190 ; first approximation.
OD 200 ;
LA 210 var     =   $2d      ; start of variables
DE 220 ptr     =   $aa      ; points to variable
PK 230 ctr     =   $ac      ; iteration counter
MJ 240 temp    =   $5c      ; temp store for flt pt #
AH 250 ;
FB 260 ; rom routines
AG 270 temf1   =   $bba2    ; unpack 5c to fac#1
JD 280 divid   =   $bb0f    ; fac#1 = var/fac#1
HP 290 plus    =   $b867    ;fac#1 = fac#2 + fac#1
DG 300 f1tem   =   $bbc7    ; pack fac#1 to 5c
IL 310 f1mem   =   $bbd4    ; pack fac#1 to memory
NI 320 print   =   $ffd2    ; chrout routine
AM 330 ;
BK 340          ldy  #$03   ;get variable's mantissa
DP 350          lda  (var),y ;check if negative
GL 360          bmi  errorexit
NC 370          dey
JC 380          lda  (var),y ;get variable's exponent
DK 390          beq  return ;return if var = zero
BD 400          lda  var    ;points to var name
AB 410          clc
HF 420          adc  #$02   ;add 2 so ptr
IO 430          sta  ptr    ;points to variable
FC 440          lda  var + 1 ;and store it
HN 450          adc  #$00
JD 460          sta  ptr + 1
MH 470          ldy  #$0    ;fill temp with zeroes
HP 480          sty  temp + 1
DA 490          sty  temp + 2
PA 500          sty  temp + 3
LB 510          sty  temp + 4
OH 520 ;
IO 530 ; now find a first approximation
LO 540 ; to the square root
AI 550 ; exponent =  exponent/2 + 40.5
KE 560 ; mantissa found from table
HG 570          lda  (ptr),y   ;get exponent (y = 0)
GM 580          ror            ;a = a/a, pop low bit
LO 590 ; carry = 1 when exponent odd
HO 600          bcs  add       ;no flag set if odd
JB 610          ldx  #$80      ;even, so set a flag bit
PH 620          stx  temp + 1
ND 630 add      adc  #$40      ;a = exponent of 1st appr
ML 640          sta  temp      ;store exponent
BD 650          iny            ;y = 1
LN 660          lda  (ptr),y   ;mantissa of variable
EH 670          ora  temp + 1 ;if expon odd add 80
JF 680          lsr            ;shift nybble right
LI 690          lsr
FJ 700          lsr
PJ 710          lsr
LJ 720          tax
DO 730          lda  tabl,x    ;get approx from table
PK 740          sta  temp + 1 ;store it in temp mantissa
GM 750 ; now use newton's method
BN 760 ; x = (x + var/x)/2
FD 770          lda  #$04
MD 780          sta  ctr       ;set counter to 4
EA 790          lda  #<temp
LG 800          ldy  #0
IF 810          jsr  temf1     ;load fac#1 from temp
LJ 820 loop     lda  ptr
NM 830          ldy  ptr + 1
NG 840          jsr  divid     ;divide fac#1 into var
AE 850          lda  #<temp
HK 860          ldy  #0
AJ 870          jsr  plus      ;add temp to fac#1
DG 880          dec  $61       ;divide fac#1 by 2
OB 890          jsr  f1tem     ;pack fac#1 to temp
ON 900          dec  ctr       ;decrement the counter
PL 910          bne  loop      ;loop if not zero
KE 920          ldx  ptr
BD 930          ldy  ptr + 1
NA 940          jsr  f1mem     ;pack fac#1 to memory
LG 950 return   rts
KI 960 errorexit lda #$3f      ; "?"
EI 970          jsr  print     ;print "?"
AM 980          rts
HB 990 tabl     .byte 03,11,18,25,32,38,44,50
GM 1000         .byte 58,69,79,89,98,107,115,123
OM 1010end
```

# Complex Number Arithmetic For The Commodore 64

Thomas Henry
North Mankato, MN

BASIC 2.0 of the Commodore 64 supports three simple data types. These are integers, floating point numbers and strings. With the addition of a simple machine language routine, it is easy to add a fourth type: complex numbers. The routine described in this article allows the representation of complex numbers by ordered pairs, and additionally supports the four basic operations of addition, subtraction, multiplication and division.

The left–bracket, [, is used to alert the Commodore 64 that you wish to perform some complex number arithmetic. A corresponding right–bracket, ], allows the system to go back to normal floating point operation. The method by which these brackets switch the system over to complex number arithmetic is based upon Brian Munshaw's article, "A New Wedge for the Commodore 64", which appeared in The Transactor, Volume 5, Issue 6, pp. 22–25. Refer to Brian's article to learn how new commands may be added to the 64 by means of the "Error Wedge".

Before seeing how to punch this new routine into your computer, let's consider the details of representation and syntax. As mentioned above, complex numbers are represented in this system by ordered pairs. Whereas highschool algebra books might denote a complex number by a+bi, the ordered pair (a,b) is used hereinafter. The first entry is always assumed to be the real part, while the second entry is the imaginary part.

The syntax for a complex number operation is (here shown for addition):

```
[(X,Y)=(A,B)+(C,D)]
```

The left–bracket signifies the complex number mode, while the right–bracket signifies the end of this mode. The variables X and Y are set equal to the results of A+C and B+D, respectively. Similar syntax holds for the other three operations. A, B, C and D may be any valid combination of values, variables or operands, but X and Y (which hold the resulting complex number) must of course be variables only.

### Adding The New Commands To Your Computer

Program 2 shows the complete assembly language listing. While it's not necessary to understand this listing to use the system, you will still find it to be quite interesting. In particular, the source code illustrates how to manipulate variables and perform floating point operations from within machine language. Also, the source code will be of inestimable value should you desire to add some new operations (like complex number conjugation, for example). So, give the code a quick glance over to become familiar with the several novel actions it performs.

On the other hand, if you simply wish to get down to business and use the program, then Program 1 is the listing you'll want. This is a BASIC generator program which creates a working copy of the complex number routine on disk, ready for use. First, enter this program into your Commodore 64. Now run it. A checksum handler will halt the program if there are errors in the DATA statements. Proofread the program and make any corrections as needed. If you make it past the checksum detector, then the complex number routine will be written to disk under the name of "COMPLEX.ML". This is the program that you will use from now on. You may discard Program 1 now, as it has served its purpose.

### Using The Complex Number Package

At this point you can load the package into memory with:

```
load "complex.ml",8,1
```

The program will load into memory from $C000 through $C1C6. You should enter NEW at this point, then activate the routine with:

```
sys 49152
```

This engages the complex number commands, making them available for use until you shut the computer off. To become familiar with the syntax, try out some examples. Here are a couple to get you started:

```
100 input a,b
110 input c,d
120 [(e,f)=(a,b)-(c,d)]
130 print e,f
```

```
100 p=12: q=13.1
110 r=54.1: s=3.14
120 [(w,y)=(2*p,3*q)/(r/12,s)]
130 print w,y
```

In the first example, the complex numbers (A,B) and (C,D) are input. The latter is subtracted from the former and the result is printed out. In this case, all of the parts of all of the complex numbers are variables (i.e., A, B, C, D, E and F).

Example two demonstrates that the parts of the complex numbers may actually be quite complicated. Here the complex number (2*P,3*Q) is divided by (R/12,S) and the result is left in the pair of variables W and Y (W is the real part and Y is the imaginary part).

### Some Interesting Details

The real and imaginary parts of the two operands may be any valid combination of values, variables and ordinary numeric operators. Both integers and floating point numbers may be used. The real and imaginary parts of the result must be variables (since they store the final result). These too may be integer or floating point types. Be forewarned, however, that in general the product or difference of two complex numbers made up of integer components rarely lead to complex numbers with integer components. In this case, the Commodore 64 will truncate the fractional part.

All of the values of the components in the complex numbers must, of course, be numeric in type. Attempting to use a string will lead to a TYPE MISMATCH ERROR. And as is the case with integers and floating point numbers, division by zero is undefined and will yield a DIVISION BY ZERO ERROR.

The algorithms used to compute these various results are:

(a,b) + (c,d) = (a + c,b + d)
(a,b)–(c,d) = (a–c,b–d)
(a,b)*(c,d) = (a*c–b*d,a*d + b*c)
(a,b)/(c,d) = ((a*c + b*d)/(c*c + d*d),(b*c–a*d)/(c*c + d*d))

### Program 1: Creates disk file "COMPLEX.ML"

```
AG 100 rem prg file gen for complex.ml
EM 110 ch=0
EE 120 for a=49152 to 49606
DI 130 read x: ch=ch+x: next
LB 140 if ch=55928 then 160
MG 150 print"error in data statements":end
FP 160 open8,8,8,"0:complex.ml,p,w"
GI 170 print#8,chr$(0);chr$(192);
MM 180 restore
KI 190 for a=49152 to 49606
LO 200 read x: print#8,chr$(x);: next
KL 210 close 8
KP 220 rem
MA 230 data 169, 11, 141,  0,  3, 169, 192, 141
AP 240 data   1,  3,  96, 224, 11, 240,   3,  76
PA 250 data 139, 227, 32, 121,  0, 201,  91, 208
CB 260 data 246, 104, 104, 32, 115,  0,  32, 250
OE 270 data 174, 32, 139, 176, 32, 141, 173, 141
DA 280 data 168,  2, 140, 169,  2, 165,  14, 141
MD 290 data 167,  2,  32, 253, 174, 32, 139, 176
EL 300 data  32, 141, 173, 141, 171,  2, 140, 172
HC 310 data   2, 165,  14, 141, 170,  2,  32, 247
HK 320 data 174, 169, 178,  32, 255, 174, 32, 250
JC 330 data 174,  32, 138, 173, 162, 173, 160,  2
CO 340 data  32, 212, 187,  32, 253, 174, 32, 138
BP 350 data 173, 162, 178, 160,  2,  32, 212, 187
DM 360 data  32, 247, 174, 141, 193,  2,  32, 115
HH 370 data   0,  32, 250, 174,  32, 138, 173, 162
GJ 380 data 183, 160,  2,  32, 212, 187,  32, 253
LD 390 data 174,  32, 138, 173,  32, 167, 193,  32
PN 400 data 247, 174, 169,  93,  32, 255, 174, 173
JM 410 data 193,  2, 201, 170, 240,  21, 201, 171
JP 420 data 240,  17, 201, 172, 208,  3,  76,  1
PC 430 data 193, 201, 173, 208,  3,  76, 216, 192
OC 440 data  76,  8, 175,  32, 161, 193, 169, 173
FI 450 data 160,  2,  32, 135, 193, 174, 168,  2
GN 460 data 172, 169,  2, 173, 167,  2,  32, 128
MO 470 data 193,  32, 155, 193, 169, 178, 160,  2
HJ 480 data  32, 135, 193, 174, 171,  2, 172, 172
BD 490 data   2, 173, 170,  2,  32, 128, 193,  96
FB 500 data  32, 180, 191,  32, 167, 193,  32,  12
AL 510 data 188,  32, 184, 193,  32, 202, 187,  32
IJ 520 data 161, 193,  32,  12, 188,  32, 184, 193
LG 530 data 169,  87, 160,  0,  32, 103, 184, 165
DG 540 data  97, 208,  3,  76, 138, 187,  32, 202
EM 550 data 187, 169, 173, 160,  2,  32, 162, 187
DE 560 data  32,  12, 188,  32, 161, 193,  32, 184
HB 570 data 193,  32, 199, 187,  32, 155, 193,  32
GJ 580 data 184, 193, 174, 168,  2, 172, 169,  2
IK 590 data  32, 212, 187,  32, 148, 193, 169, 188
KG 600 data 160,  2,  32,  40, 186, 169,  92, 160
CO 610 data   0,  32,  80, 184,  32, 199, 187,  32
FL 620 data 148, 193, 169, 183, 160,  2,  32,  40
AI 630 data 186, 173, 168,  2, 172, 169,  2,  32
JI 640 data 103, 184, 173, 193,  2, 201, 172, 240
LJ 650 data  28,  32,  12, 188,  32, 174, 193,  32
GD 660 data 190, 193,  32,  18, 187,  32, 203, 192
LM 670 data  32, 174, 193, 169,  92, 160,  0,  32
PD 680 data  15, 187,  76, 115, 193,  32, 203, 192
DK 690 data  32, 177, 193, 174, 168,  2, 172, 169
GB 700 data   2, 173, 167,  2,  32, 128, 193,  96
AG 710 data 134,  73, 132,  74,  76, 194, 169, 174
LL 720 data 193,  2, 224, 170, 208,  3,  76, 103
DC 730 data 184,  76,  80, 184, 169, 178, 160,  2
HA 740 data  76, 162, 187, 169, 188, 160,  2, 208
GB 750 data 247, 169, 183, 160,  2, 208, 241, 162
HD 760 data 188, 160,  2,  76, 212, 187, 169,  87
IF 770 data  44, 169,  92, 160,  0,  76, 162, 187
HK 780 data  32, 190, 193,  76,  43, 186, 165, 102
HP 790 data  69, 110, 133, 111, 165,  97,  96
```

### Program 2: PAL Source Code

```
LC 100 rem complex number arithmetic pal source
LB 110 open8,8,1,"0:complex.ml
JE 120 sys700
FK 130 .opt o8
CA 140 ;
EO 150 ;*****************************************
KM 160 ;*                                       *
ID 170 ;*        complex number arithmetic      *
EN 180 ;*          for the commodore 64         *
IO 190 ;*                                       *
ON 200 ;*     +,–,* and / are all supported as  *
```

```
HP 210 ;*          well as complex numbers          *
DM 220 ;*       (represented as ordered pairs).     *
AB 230 ;*                                           *
NH 240 ;*              thomas henry                 *
EC 250 ;*                                           *
CF 260 ;*************************************
EI 270 ;
ON 280 intflg   =   $0e       ;$00 = fltpt, $80 = integer
FP 290 forpnt   =   $49       ;temp pointer in list
EK 300 facexp   =   $61       ;fltpnt accum, exponent
CF 310 facsgn   =   $66       ;fltpnt accum, sign
KM 320 argsgn   =   $6e       ;fltpnt accum #2, sign
HD 330 arisgn   =   $6f       ;fac1 & fac2 sign cmp flag
DL 340 chrget   =   $73       ;basic character get rtn
OL 350 chrgot   =   $79       ;re-get old character
FJ 360 realot   =   $02a7     ;real part output pnt
IF 370 imagot   =   realot+3  ;img part output pnt
IO 380 real1    =   imagot+3  ;1st number real part
MP 390 imag1    =   real1+5   ;1st number img part
JM 400 real2    =   imag1+5   ;2nd number real part
FP 410 imag2    =   real2+5   ;2nd number img part
GJ 420 cmxop    =   imag2+5   ;complex # operator
ON 430 ierror   =   $0300     ;vector to error routine
BM 440 varstr   =   $a9c2     ;store numeric variable
CN 450 frmnum   =   $ad8a     ;evaluate a numeric
CH 460 chknum   =   $ad8d     ;check for number
LP 470 chkcls   =   $aef7     ;check for close paren
DB 480 chkopn   =   $aefa     ;check for open paren
BC 490 chkcom   =   $aefd     ;check for comma
DI 500 chkchr   =   $aeff     ;check for chr in acc
FD 510 snerr    =   $af08     ;do 'syntax error'
ID 520 ptrget   =   $b08b     ;find variable
EJ 530 fsub     =   $b850     ;fac1 = (a:lsb,y:msb) - fac1
KD 540 fadd     =   $b867     ;fac1 = (a:lsb,y:msb) + fac1
GP 550 faddt    =   $b86a     ;fac1 = fac2 + fac1 (setup)
MF 560 fmult    =   $ba28     ;fac1 = (a:lsb,y:msb) * fac1
LL 570 fdiv     =   $bb0f     ;fac1 = (a:lsb,y:msb) / fac1
GK 580 fdivt    =   $bb12     ;fac1 = fac2 / fac1
DC 590 divzer   =   $bb8a     ;'division by zero'
KJ 600 movfm    =   $bba2     ;fac1 = (a:lsb,y:msb)
IK 610 flt5c    =   $bbc7     ;read fac1 to $5c-$60
NM 620 flt57    =   $bbca     ;read fac1 to $57-$5b
FD 630 movmf    =   $bbd4     ;(x:lsb,y:msb) = fac1
NN 640 movaf    =   $bc0c     ;fac2 = fac1
DH 650 negop    =   $bfb4     ;fac1 = -fac1
FM 660 olderr   =   $e38b     ;continue old error routine
EB 670 ;
JJ 680 *= $c000
IC 690 ;
HI 700 init     lda  #<cmxrtn ;change error vector
NC 710          sta  ierror
KO 720          lda  #>cmxrtn
IC 730          sta  ierror+1
AN 740          rts
EG 750 ;
AN 760 cmxrtn   cpx  #$0b     ;"syntax-error?
GE 770          beq  cmx1     ;yes, move on.
FD 780 cmx0     jmp  olderr   ;no, do regular error routine
MI 790 ;
EA 800 cmx1     jsr  chrgot   ;reget offending character
MF 810          cmp  #'[      ;"left-bracket?
EC 820          bne  cmx0     ;no, ordinary error
IP 830          pla           ;pop junk from stack
AN 840          pla
FI 850          jsr  chrget   ;fetch next character
KA 860          jsr  chkopn   ;check for open paren
BH 870          jsr  ptrget   ;find real output
OM 880          jsr  chknum   ;check if numeric
AP 890 ;
GI 900          sta  realot+1 ;lsb
GP 910          sty  realot+2 ;msb
JF 920          lda  intflg   ;save integer flag
OA 930          sta  realot
FK 940          jsr  chkcom   ;check for comma
CO 950          jsr  ptrget   ;find img output
OB 960          jsr  chknum   ;check if numeric
BC 970          sta  imagot+1 ;lsb
PI 980          sty  imagot+2 ;msb
HI 990          lda  intflg
BG 1000           sta  imagot   ;save integer flag
IH 1010           jsr  chkcls   ;check for close paren
CA 1020           lda  #$b2     ;check for '='
HI 1030           jsr  chkchr
OL 1040           jsr  chkopn   ;check for open paren
PA 1050           jsr  frmnum   ;evaluate real #1
HO 1060           ldx  #<real1
BP 1070           ldy  #>real1
MN 1080           jsr  movmf    ;save it for later
LD 1090           jsr  chkcom   ;check for comma
PL 1100           jsr  frmnum   ;evaluate img #1
EB 1110           ldx  #<imag1
OB 1120           ldy  #>imag1
OA 1130           jsr  movmf    ;save it for later
KP 1140           jsr  chkcls   ;check for close paren
PB 1150           sta  cmxop    ;save operator
PH 1160           jsr  chrget   ;get next character
AE 1170           jsr  chkopn   ;check for open paren
EJ 1180           jsr  frmnum   ;evaluate real #2
MG 1190           ldx  #<real2
GH 1200           ldy  #>real2
OF 1210           jsr  movmf    ;save it for later
NL 1220           jsr  chkcom   ;check for comma
DE 1230           jsr  frmnum   ;evaluate img #2
DA 1240           jsr  otimg2   ;save it for later
IG 1250           jsr  chkcls   ;check for close paren
DN 1260           lda  #']      ;check for ']'
HH 1270           jsr  chkchr
GH 1280 ;
FF 1290           lda  cmxop    ;restore operator
HA 1300           cmp  #$aa     ;"is it + ?
DB 1310           beq  cmx4     ;branch if it is
OJ 1320 ;
PC 1330           cmp  #$ab     ;"is it - ?
BD 1340           beq  cmx4     ;branch if it is
ML 1350 ;
IE 1360           cmp  #$ac     ;"is it * ?
CM 1370           bne  cmx2     ;branch if not
EO 1380           jmp  cmx8
EO 1390 ;
CI 1400 cmx2      cmp  #$ad     ;"is it / ?
OO 1410           bne  cmx3     ;branch if not
EA 1420           jmp  cmx6
MA 1430 ;
NP 1440 cmx3      jmp  snerr    ;'syntax error'
AC 1450 ;
FM 1460 cmx4      jsr  inrel2   ;bring in real #2
FC 1470           lda  #<real1
LI 1480           ldy  #>real1
```

```
PK 1490        jsr  addsub   ;do + or -.
IC 1500        ldx  realot+1
KD 1510        ldy  realot+2
KE 1520        lda  realot   ;integer flag
BM 1530        jsr  storfp   ;store real result
CD 1540        jsr  inimg2   ;bring in img #2
AH 1550        lda  #<imag1
GN 1560        ldy  #>imag1
BN 1570        jsr  addsub   ;do + or -
PP 1580 cmx5   ldx  imagot+1
OA 1590        ldy  imagot+2
JP 1600        lda  imagot   ;integer flag
BP 1610        jsr  storfp   ;store img result
AE 1620        rts
EN 1630 ;
AG 1640 cmx6   jsr  negop    ;negate d for conjugate
ND 1650        jsr  otimg2   ;save d
BK 1660        jsr  movaf    ;fac2 = d
GA 1670        jsr  cmult    ;fac1 = d*d
AM 1680        jsr  flt57    ;save d*d
PF 1690        jsr  inrel2   ;bring in c
FM 1700        jsr  movaf    ;fac2 = c
IC 1710        jsr  cmult    ;fac1 = c*c
GA 1720        lda  #$57
FE 1730        ldy  #$00
AE 1740        jsr  fadd     ;fac1 = c*c + d*d
KA 1750        lda  facexp   ;check for zero
AM 1760        bne  cmx7     ;branch if okay
IH 1770        jmp  divzer   ;'division by zero'
KG 1780 ;
OG 1790 cmx7   jsr  flt57    ;save c*c + d*d
EH 1800 cmx8   lda  #<real1
FN 1810        ldy  #>real1
FH 1820        jsr  movfm    ;bring in a
PD 1830        jsr  movaf    ;fac2 = a
FP 1840        jsr  inrel2   ;bring in c
MK 1850        jsr  cmult    ;fac1 = a*c
EL 1860        jsr  flt5c    ;save it
GB 1870        jsr  inimg2   ;bring in d
MM 1880        jsr  cmult    ;fac1 = a*d
OK 1890        ldx  realot+1
AM 1900        ldy  realot+2
MB 1910        jsr  movmf    ;save a*d
AE 1920        jsr  inimg1   ;bring in b
PO 1930        lda  #<imag2
FF 1940        ldy  #>imag2
JB 1950        jsr  fmult    ;fac1 = b*d
GC 1960        lda  #$5c
FD 1970        ldy  #$00
FH 1980        jsr  fsub     ;fac1 = a*c - b*d
EN 1990        jsr  flt5c    ;save it back
AJ 2000        jsr  inimg1   ;bring in b
EE 2010        lda  #<real2
KK 2020        ldy  #>real2
HG 2030        jsr  fmult    ;fac1 = b*c
IO 2040        lda  realot+1
GF 2050        ldy  realot+2
EH 2060        jsr  fadd     ;fac1 = a*d + b*c
CF 2070        lda  cmxop    ;check operator
IB 2080        cmp  #$ac     ;"is it *?
IO 2090        beq  cmx9     ;yes, branch
JL 2100        jsr  movaf    ;fac2 = a*d + b*c
KD 2110        jsr  flf57    ;fac1 = mag squared
JE 2120        jsr  adjsgn   ;adjust sign
GF 2130        jsr  fdivt    ;fac1 = (a*d+b*c)/(c*c+d*d)
KH 2140        jsr  cmx5     ;store imaginary part
CG 2150        jsr  flf57    ;fac1 = mag squared
AG 2160        lda  #$5c     ;fac1 = (a*c-b*d)/(c*c+d*d)
NP 2170        ldy  #$00
IE 2180        jsr  fdiv     ;final result
CC 2190        jmp  cmx10
OA 2200 ;
GM 2210 cmx9   jsr  cmx5     ;store imaginary part
DG 2220        jsr  flf5c    ;fac1 = real part
GJ 2230 cmx10  ldx  realot+1
EB 2240        ldy  realot+2
EC 2250        lda  realot   ;integer flag
OE 2260        jsr  storfp   ;store real part
KM 2270        rts
OF 2280 ;
AE 2290 ;/// complex number utilities ///
CH 2300 ;
FO 2310 storfp stx  forpnt
NH 2320        sty  forpnt+1
HK 2330        jmp  varstr   ;go store value
KJ 2340 ;
FO 2350 addsub ldx  cmxop    ;check operator
EF 2360        cpx  #$aa     ;"is it + ?
OD 2370        bne  as1
OC 2380        jmp  fadd     ;go do +
MM 2390 ;
OE 2400 as1    jmp  fsub
AO 2410 ;
GH 2420 inimg1 lda  #<imag1  ;bring in imag1
MD 2430        ldy  #>imag1
LP 2440 cmovfm jmp  movfm
IA 2450 ;
EK 2460 inimg2 lda  #<imag2  ;bring in imag2
HG 2470        ldy  #>imag2
NK 2480        bne  cmovfm   ;branch always
AD 2490 ;
OP 2500 inrel2 lda  #<real2  ;bring in real2
EJ 2510        ldy  #>real2
FN 2520        bne  cmovfm   ;branch always
IF 2530 ;
HL 2540 otimg2 ldx  #<imag2  ;store imag2
HL 2550        ldy  #>imag2
NE 2560        jmp  movmf    ;branch always
AI 2570 ;
BD 2580 flf57  lda  #$57
JP 2590        .byte $2c
JG 2600 flf5c  lda  #$5c
FL 2610        ldy  #$00
AC 2620        jmp  movfm    ;move to fac1
ML 2630 ;
MH 2640 cmult  jsr  adjsgn   ;set up for multiply
JL 2650        jmp  fmult+3  ;fac1 = fac1 * fac2
KN 2660 ;
GB 2670 adjsgn lda  facsgn   ;adjust sign
CB 2680        eor  argsgn
IL 2690        sta  arisgn
JH 2700        lda  facexp
CI 2710        rts
GB 2720 ;
GI 2730 .end
```

# FAC1 Facts

John Houghton
Collingwood, Ontario

QUESTION: *In machine language, how do you multiply 3 × 4 ?*

ANSWER: *Well, you take the multiplicand (3) and add it to itself, the number of times in the multiplier (4).*

QUESTION: *OK! Then how do you multiply 10.125 x 6.375 ?*

ANSWER: *Sorry, anything past integer arithmetic in M.L. gets left alone. Either do it in Basic or not at all.*

"The time has come" the walrus said, "to think of many things. Like. . ." floating point arithmetic.

Remember way back when, in high school algebra, something called logarithms was discussed? Well, floating point notation is similar in nature. Instead of base 10 notation, F.P. on the computer uses base 2. Therefore, a number is the equivalent of the Mantissa times 2 to the power of the Exponent (In simplified terms). Sounds confusing, and it can be, especially when it is time to explain how to add, multiply etc., but exact knowledge of how F.P. works is not needed to be able to use it in a machine language program. In fact the best advice is to cheat and let ye olde trusty micro-processor do the work. All that is needed to be known are the ground rules by which to operate.

The information discussed in this article pertains directly to the Commodore 64 but other Commodore machines will have similar routines in their ROM, just check the ROM memory maps. After a discussion of these ROM routines, a short PAL source code program will run through some of the operations.

There are two floating point accumulators in the 64. FAC1 at $61 to $65 and FAC2 at $69 to $6e. $61 and $69 contain the exponents, $62 to $65 and $6a to 6d contain the 4 byte mantissa and $66 and $6f contain the signs for each number. There is a main work/storage area at $57 to $60 used for F.P. crunching. As well there are some miscellaneous but important locations at $67/$68 and $6f to $72.

Now here is a list of some of the routines for manipulating floating point numbers.

**$bcf3: convert ascii string to FAC1**

-To use this routine place an ascii numeric string, with a 0 after the string, in a buffer. Point the CHRGET pointers at $7a/$7b to the first character and call CHRGOT before entry.
-This routine is called by $ad9e and $ae83 routines.
-It clears FAC1, FAC2 and numeric work area $5d to $60 before converting the string to F.P. notation. Therefore preserve anything in these three areas that are needed later.
-It works like the VAL function in basic, indeed VAL calls this routine, and will ignore alphabetic entry after an ascii numeric string.
-It will accept scientific notation incorporating an 'e' and a leading '+' or '-'.

**$bbc7: move FAC1 to memory.**

-This routine has several entry points depending on where FAC1 is to be stored.
-Enter at $bbc7 and FAC1 goes to $5c-$60 (and will get clobbered if $bcf3 is called again).
-Enter at $bbca and FAC1 goes to $57-$5b (and is immune to $bcf3 calls).
-Enter at $bbd4 and FAC1 goes to the address pointed to by the .X (low) and .Y (high) registers.

**$bba2: move memory to FAC1.**

-Point the .A (low) and .Y (high) registers at memory and this routine loads the 5 byte floating point number into FAC1.

**$ba8c: move memory to FAC2.**

-Same as above for FAC2.

**$bc0c: round and move FAC1 to FAC2.**
**$bc0f: move FAC1 to FAC2 no rounding.**
**$bbfc: move FAC2 to FAC1**
**$bc1b: round FAC1**

**$bc5b: compare FAC1 to memory.**

-Point the .A (low) and .Y (high) registers to an F.P. number in memory, and this routine will return a 0 in .A if they are the same, a 1 in .A if FAC1 is greater than memory and $ff in .A if memory is greater than FAC1.

**$b7f7: convert FAC1 to 2 byte integer.**

-.A and .Y as well as $14/$15 contain the integer.
(Note: this only works for positive values. If you know the value in FAC1 is positive, set the sign bit at $66 to zero. Otherwise, use $b1bf to convert to a signed integer, or $bc9b to get a 4 byte signed integer.)

**$bc9b: convert FAC1 to 4 byte integer.**

-stored in FAC1 mantissa area $62 to $65.

**$bddd: convert fac1 to ascii string.**

-The ascii string is stored at $0100 with a 0 after the string and the .A (low) and .Y (high) registers pointing to the string address. In the process, FAC1 is altered as well as memory area $5d and $5e. To print the string after this routine make a direct call to $ab1e.

The next routines need the address of the floating point number stored in the .A and .Y registers before being called. They all move that number to FAC2 and then perform the appropriate operation.

**$b850: subtract memory form FAC1**
**$b867: add memory to FAC1**
**$ba28: multiply FAC1 by memory**
**$bb0f: divide memory by FAC1**

An alternative to these arithmetic operations is to move the needed number to FAC2 by one of the previously listed move routines. Then

for the add and divide routines load the A or X register with $61, and call one of the following:

**$b853: subtract FAC2 from FAC1**
**$b86a: add FAC2 to FAC1**
**$ba30: multiply FAC1 by FAC2**
**$bb12: divide FAC2 by FAC1**
**$bf7b: raise FAC2 to the power of FAC1**

Other useful routines are listed below.

**$b849: add .5 to FAC1 (no pointers needed)**
**$bccc: perform INT (calls $bc9b and reconverts to F.P.)**
**$bae2: multiply FAC1 by 10.**
**$bafe: divide FAC1 by 10.**
**$e264: perform COS to FAC1.**
**$e26b: perform SIN to FAC1.**
**$e2b4: perform TAN to FAC1.**
**$e30e: perform ATN to FAC1.**

The trigonometry functions above need the angle, expressed in radians, stored in FAC1.

And there are some five byte floating point constants stored in ROM that you might want to use.

**$bf11:** constant .5
**$b9bc:** constant 1
**$baf9:** constant 10
**$aea8:** constant pi
**$e2e0:** constant pi/2
**$e2e5:** constant 2*pi
**$e2ea:** constant 1/4

There are some other routines in ROM that perform overflow checking, exponent addition, and sign checking that will need to be accessed when getting into long and complex formulas but for now the routines outlined will get the fear of using floating point numbers from a machine language program out of the way.

Now onto a short discussion of the program. First, BUFFR1 is converted from an ascii string to an F.P. number in FAC1 and moved to memory location $57. Then BUFFR2 is similarly converted and stored at $5c. The two numbers are then multiplied and stored in $57. Finally the two original numbers are added together and a check to see if the sum is less than the product. If it is less, then keep adding until the sum exceeds the product. The program will then print the sum to the screen, print the product to the screen and print the number of times it added the two numbers together.

Here are some alterations to the program. Try messing around with the ascii numbers and see what other values will be produced. Change the arithmetic operations in lines 510 or 590. Maybe use the move routine at $bc0c in line 600 instead, to see what effect takes place on a lot of additions. Use large original numbers to find out when scientific notation displays take over.

That's probably enough to make the old wood start burning, but it will be interesting to learn who will be the first to convert Chris Zamara's SPRITE ROTATE from VOL. 5 ISS. 1 entirely into M.L.???

```
DA  100 rem ••• fac1 facts •••
MG  110 rem ••• by john houghton •••
MG  120 rem ••• collingwood ont •••
KK  140 rem
ED  150 rem pal source code
LH  170 sys700
GC  190 .opt oo
OD  200 ;
DM  210 * = $c000
CF  220 ;
AO  230 ;a routine to demonstrate the use
CN  240 ;of floating point numbers, the
NO  250 ;associated floating point
LB  260 ;accumulators and rom routines.
EI  270 ;
LD  280       lda  #$00
OB  290       sta  count     ;clear integer
HH  300       sta  count+1 ;storage
BN  310       lda  $7a       ;store chrget
BB  320       sta  getlo     ;pointers
KA  330       lda  $7b
GA  340       sta  gethi
CI  350       lda  #<buffr1 ;point chrget to
FH  360       sta  $7a       ;1st number
FA  370       lda  #>buffr1
KH  380       sta  $7b
IH  390       jsr  $0079     ;call chrgot
JB  400       jsr  $bcf3     ;ascii to fac1
BJ  410       jsr  $bbca     ;fac1 to mem $57
MM  420       lda  #<buffr2 ;point chrget to
CI  430       sta  $7a       ;2nd number
PE  440       lda  #>buffr2
AM  450       sta  $7b
OL  460       jsr  $0079     ;call chrgot
PF  470       jsr  $bcf3     ;ascii to fac1
JN  480       jsr  $bbc7     ;fac1 to mem $5c
ID  490       lda  #$57
HH  500       ldy  #$00
PH  510       jsr  $ba28     ;$57 to fac2 & mult
LD  520       jsr  $bbca     ;product to $57
II  530 ;
EI  540 add   =    *
EK  550       lda  #$5c
DL  560       ldy  #$00
PG  570       jsr  $bba2     ;$5c to fac1
NN  580       ldx  $61       ;exponent fac1
JG  590       jsr  $b86a     ;add fac2 to fac1
JH  600       jsr  $bc0f     ;sum to fac2
FD  610       inc  count     ;inc number of
IF  620       bne  nex       ;additions
BF  630       inc  count+1
GP  640 ;
PF  650 nex   =    *
CO  660       lda  #$57
BC  670       ldy  #$00
BN  680       jsr  $bc5b     ;compare sum & prod
NI  690       bmi  add       ;fac1 < $57
JC  700       jsr  $bddd     ;fac1 to ascii
MI  710       jsr  $ab1e     ;print sum
DE  720       lda  #$0d
DH  730       jsr  $ffd2
CD  740       lda  #$57
BH  750       ldy  #$00
BC  760       jsr  $bba2     ;$57 to fac1
PG  770       jsr  $bddd     ;fac1 to ascii
CN  780       jsr  $ab1e     ;print product
JI  790       lda  #$0d
JL  800       jsr  $ffd2
ED  810       lda  count+1 ;get count of
NA  820       ldx  count     ;additions convert
PC  830       jsr  $bdcd     ;int to ascii
CB  840       jsr  $ab1e     ;print count
AO  850       lda  getlo
GB  860       sta  $7a       ;restore chrget
KN  870       lda  gethi
OG  880       sta  $7b
GG  890       rts
KP  900 ;
NK  910 count * = * +2
HO  920 getlo * = * +1
NM  930 gethi * = * +1
CC  940 ;
PP  950 buffr1 .asc "10.125"
AI  960        .byte $00
AE  970 ;
DN  980 buffr2 .asc "6.375"
OJ  990        .byte $00
```

# High–Speed Integer Multiplies and Divides

**Donald A. Branson**
**St. Louis, Missouri**

*. . .If your processor has multiply/divide commands, no problem. However, if it does not. . .*

If you have written any amount of assembler code, you have probably encountered a situation where you wanted to do a multiplication or a division. If your processor has multiply/divide commands, no problem. However, if it does not (as in the case of the 6502–series CPU in Commodore's 8–bit computers), you may have done a multiply by adding an operand to itself, decrementing the operand at the same time until it reaches zero.

This is probably the most straightforward way to do a multiply, but a binary multiply can be done the same way a person multiplies a decimal number by hand. This way is as easy to do and, in all but a few cases, is quite a bit faster.

**Multiplication**

First, let's look at the way a decimal multiply is done, in order to draw attention to some key elements in any multiply. The procedure is expanded on the right to show more specifically what is happening here.

```
  35      1)   5 * 5   =   25
x 15      2)   5 * 30  =  150
 175      3)  10 * 5   =   50
 350      4)  10 * 30  =  300
 525                      525
```

It can be seen that 35 has not been added to itself 15 times or 15 added to itself 35 times. Doing it that way would take longer and, if you're doing it by hand, more prone to error.

What has been done is to multiply each digit in the first number against all the digits in the other number.

This same method can be applied to binary multiplication.

In order to look at one binary digit (bit) at a time, a combination of the LSR and ROR operations is used to shift the bit into the carry bit, where it can be examined with a BCC or BCS operation. In this way, one digit can be multiplied at a time. If this were a decimal multiply, we would have to have a table of results for each possible combination of one digit multiplies, like this:

```
1 * 1 = 1
1 * 2 = 2
  .
  .
2 * 1 = 2
2 * 2 = 4
  .
  .
```

When we multiply decimal, we have this table in our heads. However, the table is simplified when we do binary multiplication. If fact, it is simplified to the point that the result of a multiply can be one of two things: a zero bit or a one bit.

A result table does not need to be kept for binary multiplication. The table is effectively implied in the operands.

For example:

```
15 = $0F = %0000 1111
35 = $23 = %0010 0011

        00001111    M1
      x 00100011    M2
        00001111
       00001111
      00000000
     00000000
    00000000
   00001111
  00000000
 00000000
 0000001000001101    R
  ($020D = 525)
```

Now to look at the process in a way that can be easily represented in a computer. We'll call the multiplicand M1, the multiplier M2, and the field where the result is left R, like this:

```
M1: 0000 0000 0000 1111
M2: 0000 0000 0010 0011
R:  0000 0000 0000 0000
```

Since the result will be sixteen bits (525 > 255), all the fields are represented as sixteen bits.

First, M1 is shifted right, so that each bit can be looked at from right to left as in the decimal multiply shown before. As you can see in the example of binary multiplication above, the digit can be either one or zero, and the result of the single-digit multiply is either zero, or a duplicate of M2, properly shifted to the left. After each digit in M1 is shifted into the carry bit, M2 is added to R if the carry is one, then M2 is shifted left, in preparation for the next digit.

Here, 'SR' stands for shift right, 'SL' for shift left.

```
pass                                          carry
 1-      SR M1:  0000 0000 0000 0111    1
      R=R+M2:  0000 0000 0010 0011
         SL M2:  0000 0000 0100 0110

 2-      SR M1:  0000 0000 0000 0011    1
      R=R+M2:  0000 0000 0110 1001
         SL M2:  0000 0000 1000 1100

 3-      SR M1:  0000 0000 0000 0001    1
      R=R+M2:  0000 0000 1111 0101
         SL M2:  0000 0001 0001 1000

 4-      SR M1:  0000 0000 0000 0000    1
      R=R+M2:  0000 0010 0000 1101
         SL M2:  0000 0010 0011 0000
```

Note that we can stop here, because M1 is zero, and the carry bit will no longer be set as a result of shifting M1, and therefore M2 will not be added to R any more times. At this point, the result of the multiply is left in R.

Here is how this process looks in assembler:

```
MLOOP EQU *          ;in PAL- MLOOP = *
      LSR  M1        ;SR M1
      ROR  M1+1      ;the carry from M1 → M1+1
      BCC  SHIFT     ;add on carry set
      CLC            ;R=R+M2
      LDA  M2+1
      ADC  R+1
      STA  R+1
      LDA  M2
      ADC  R
      STA  R
SHIFT EQU *
      ASL  M2+1      ;SL M2
      ROL  M2
      LDA  M1+1
      BNE  MLOOP
      LDA  M1        ;if M1 and M1+1=0
      BNE  MLOOP     ;when we are done
      RTS
M1    HEX  0023      ;or .BYTE $00,$23
M2    HEX  000F      ;or .BYTE $00,$0F
R     HEX  0000      ;or .BYTE $00,$00
```

**Division**

Now we'll take a look at division. Binary division can also be done by mimicking the way decimal division is done. Let's look at an example of how to compute 43/5, giving a result of 8 remainder 3. Here's how it would be done by hand, in decimal.

```
      8    r 3
   ------
5 ) 43
    40
    --
     3
```

1) 8 * 5 = 40
2) 43 - 40 = 3

M1 = 5 = 0000 0101  M2 = 43 = 0010 1011
R1 = 0000 0000  R2 = 0000 0000

In the case of division, two result fields are needed, one for the quotient, the other for the remainder. Specifically:

M1 / M2 = R1 rem R2

M2 will be shifted left into R2, and R2 will then be compared to M1. If R2 is greater than or equal to M1, then M1 will fit into R2. The result of a binary divide can be only a zero or a one, again saving us the trouble of having a result table, as is necessary for decimal division. If M1 will fit into R2, then we subtract M1 from R2, leaving the result in R2. Finally, R1 is shifted left, with the carry bit from the compare going into the least significant bit.

```
pass
 1-  SL M2 into R2   R2 = 0000 0000   M2 = 0101 0110
     CMP R2-M1 (carry clear)
     SL R1           R1 = 0000 0000

 2-  SL M2 into R2   R2 = 0000 0000   M2 = 1010 1100
     CMP R2-M1 (carry clear)
     SL R1           R1 = 0000 0000

 3-  SL M2 into R2   R2 = 0000 0001   M2 = 0101 1000
     CMP R2-M1 (carry clear)
     SL R1           R1 = 0000 0000

 4-  SL M2 into R2   R2 = 0000 0010   M2 = 1011 0000
     CMP R2-M1 (carry clear)
     SL R1           R1 = 0000 0000

 5-  SL M2 into R2   R2 = 0000 0101   M2 = 0110 0000
     CMP R2-M1 (carry clear)
     SL R1           R1 = 0000 0001
     R2 = R2-M1      R2 = 0000 0000

 6-  SL M2 into R2   R2 = 0000 0000   M2 = 1100 0000
     CMP R2-M1 (carry clear)
     SL R1           R1 = 0000 0010

 7-  SL M2 into R2   R2 = 0000 0001   M2 = 1000 0000
     CMP R2-M1 (carry clear)
     SL R1           R1 = 0000 0100
```

*(. . .cont'd on page 45)*

# Secondary Address Bits

**Jim Butterfield**
**Toronto, Ontario**

*. . .a characteristic of secondary addresses that is not well known. . .*

When you work a device whose number is 4 or more, you are allowed (sometimes required) to use an extra number called a *secondary address*. The purpose of the secondary address is to choose a channel *within* the device. In other words, if I'm using the disk (usually device 8), I may want to have several things going at one time: one file might be in the process of being read, with another separate file being written at the same time. Although these are happening in the same *device* (the disk, number 8), we want to keep them separate. To do this, we supply unique secondary addresses for each process.

This article documents a characteristic of secondary addresses that is not well known, plus an oddity in the way it is sometimes used. The characteristic is this: Commodore Basic always adds 96 to the SA (secondary address) used by an OPEN command before storing this value. Machine language routines, however, don't do this, and you may want to add in the value to keep operation consistent. The oddity is this: despite documentation, the number you supply to go with a LOAD command is *not* a secondary address. It's just a flag, and will later be replaced by a SA of 0 (plus the extra 96).

**Try It**

Type the following direct Basic command on your computer:

```
OPEN 2,8,15
```

. . .now you can check the three values you have supplied by PEEK–ing them directly from the tables where they have been stored. Assuming this is your only active file, you should see the three values at:

| | |
|---|---|
| VIC–20, Commodore 64: | 601, 611, 621 |
| Most PET/CBM: | 593, 603, 613 |
| B–128 | 820, 830, 840 |
| C–16, Plus–4 | 1289, 1299, 1309 |
| Commodore 128 | 866, 876, 886 |

Using the PEEK address from your machine, if you PEEK the first address, you'll see the value 2 from the command that was entered. PEEK the second value and you'll see the 8. . . but when you PEEK the third value, the 15 has mysteriously changed to 111. The computer has added 96. Why?

I don't know. . . but I do know that the Basic side of the computer is quite specific about doing this. And if there is no secondary address, the Basic handler carefully sets the SA to a value of 255. . . a flag value which signals to the various parts of the kernal that there is no secondary address.

If a secondary address is being used, it's sent out every time we wish to connect to a file. When we call the device (usually the disk drive) we signal which sub–channel we are using via the SA. And there's more: the SA is also used to signal an OPEN or CLOSE as well as a normal data transmission.

Here's the system: When we open a file, we send the original SA, plus 240, to the device, followed by the file name. When the disk sees this value (240 to 255), it knows to expect a file name to follow – not data. When we close a file, we send the original SA, plus 224, to the device. When the disk sees this value (224 to 239), it knows to close the file. Finally, when we send or receive data, we send the SA as it is stored in memory (which includes the added value of 96). The disk sees that the SA is neither an OPEN nor a CLOSE, and handles the data as desired.

Now. . . when Basic puts the SA away for you, all the above happens automatically (add 96 to value supplied, or set 255 if there is no secondary address). But if you do it yourself in machine language, you must do it all.

So. . . if you're going to call SETLFS as a first step towards calling OPEN, set up the secondary address correctly. In other words, if you were planning to OPEN in a similar manner to the above command (OPEN2,8,15), I'd suggest you code:

```
LDA #$02
LDX #$08
LDY #$6F
JSR $FFBA ;(SETLFS)
```

Note that the secondary address of 15 (hex #$0F) has had a value of 96 (hex $60) added to it, to make a total of $6F.

And if you want to open to the printer with the equivalent of OPEN3,4. . . I recommend:

```
LDA #$03
LDX #$04
LDY #$FF
JSR $FFBA ;(SETLFS)
```

To signal that no secondary address is called for, put 255 (hex FF) into the Y register.

Both of the above calls should be followed soon by a call to OPEN ($FFC0). Because here's the catch: the above setup does NOT work as expected if you're going to call LOAD ($FFD5).

The documentation doesn't tell you this. . . but if you are going to make a call to LOAD, your prior call to SETLFS does **not** set a secondary address. Instead, it sets a flag. A value of binary zero allows the program to relocate as it is loaded; any other (non-zero) value triggers a load to a fixed address, with no relocation. This last is what is most often wanted.

And in the case of LOAD, you do NOT want to add 96. If your objective is relocation, the only value that works in the Y register is zero. . . add 96 and it isn't zero any more, and you will NOT relocate.

I don't use calls to LOAD much. In most cases, it seems to me to be as easy to OPEN and read the bytes myself, stacking them away under program control after examining them. But if you wish to use LOAD, go ahead. . . just remember that the SETLFS secondary address is really not a secondary address at all.

One other related subject, to do with relative files. If you want to position to a particular record in a file, you must use the secondary address of that file in the "P" command. Again, I recommend that you use an added 96 as part of the value. Thus, if you have coded:

```
OPEN 15,8,15
OPEN 1,2,3,"0:RELDATA,L,"+CHR$(75)
```

. . .opening a file, then to position to record number five, I'd suggest:

```
PRINT#15,"P"+CHR$(99)+CHR$(5)+CHR$(0)+CHR$(1)
```

The secondary address for the file is three, as shown in the OPEN 1,2,3. . . statement. But I'd suggest that your position command add 96 as shown to specify a secondary address of 99.

Do you need to carefully do all this, every time? I honestly don't know for sure. Sometimes everything seems to work when you don't worry about the extra bits. But when you carefully trace Basic code (especially the 4.0 and 7.0 versions), you see that Commodore always puts these extra values in. Maybe they know something.

And with the wide variety of disk drives and programs, you can never be sure when you might come up with a combination that needs those extra bits to be exactly right. I don't know about you. . . but my data files are too important for me to take any unnecessary chances.

*(Cont'd from page 43)*

8- SL M2 into R2 R2 = 0000 0011 M2 = 0000 0000
CMP R2-M1 (carry clear)
SL R1 R1 = 0000 1000

Now we can stop, because all eight bits of M2 have been processed.

Here is how this process looks in assembler. To compare R2 and M1, a subtract is used. The result of this subtract is saved in registers X and Y, so that the subtract does not have to be repeated, thus saving a few cycles.

```
DLOOP    EQU *
         ASL  M2+1
         ROL  M2
         ROL  M2+1
         ROL  R2
         SEC
         LDA  R2+1
         SBC  M1+1
         TAX             ;save low byte
         LDA  R2
         SBC  M1
         TAY             ;save high byte
         BCC  SKIPSAVE
         STX  R2+1       ;store saved bytes
         STY  R2         ;in R2
SKIPSAVE EQU *
         ROL  R1+1       ;shift carry from
         ROL  R1         ;subtract into R1
         LDA  M2+1
         BNE  DLOOP
         LDA  M2
         BNE  DLOOP
         RTS
M1       HEX  0005
M2       HEX  002B
R1       HEX  0000
R2       HEX  0000
```

**Conclusion**

To make these routines even faster, use zero page memory for all the fields. They will also take less memory this way.

Where will these routines come in handy? Well, for starters, here are some ideas.

- Graphics programs, for calculating the position of a dot, plotting a line, or drawing a circle, or drawing a circle.
- Compilers, for calculating memory needed for arrays, or for finding a specific element in an array. Also good for fast integer multiply/divides.
- Calculating prime numbers.

There are many places where multiplies or divides are used, and anywhere one is used, one of these routines can make your code from a few times to hundreds of times faster.

# Interfacing and Controlling the Armatron Robot

**J.J. Barbarello**
**Englishtown, NJ**

*. . .the original Armatron from Radio Shack was both agony and ecstacy. .*

For those of us interested in experimenting with robotics, the original Armatron from Radio Shack was both agony and ecstacy. Still available, armatron is a low cost, fully functioning robot arm with six degrees of freedom (meaning there are six different portions of the arm whose movement can be controlled). For robotics experimentation its serious shortcomings include the inability to lift anything of measurable weight and its totally mechanical controls. While many different articles appeared describing modifications to allow computer linkup of Armatron, all (including one I did) required a substantial amount of mechanical skill.

These problems have been overcome in a newer version called Mobile Armatron. This under $40.00 robot has enough power to lift items in the one pound range while retaining six degrees of freedom. Best of all, it is electrically controlled. As its name implies the robot can move on its two wheels. On the negative side, the elbow joint is not controllable (but you can manually move it).

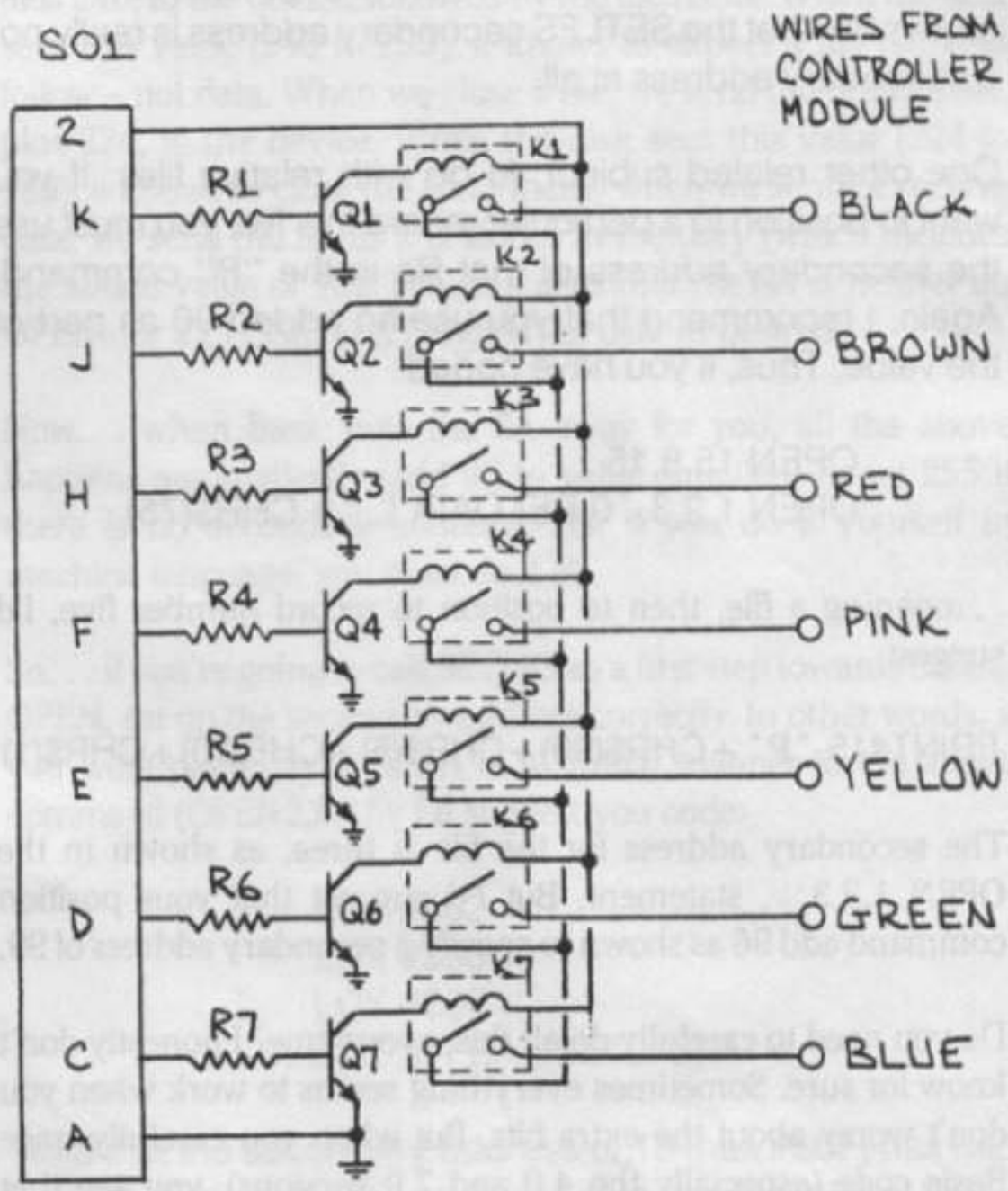
If the thought of experimenting with a computer controlled robotics device is appealing, this project is for you. The project consists of a simple interface and a comprehensive control program for the C–64. Including the cost of the robot, the total project cost should be around $60.00.

## The Interface

Mobile Armatron has a control module that is simply a series of switches. It can select one of four directions (forward, back, left and right), arm up, arm down, wrist up, wrist down, wrist turn and fingers open/clamp. The robot is powered by four D size batteries that provide plus and minus three volts for the internal motors. By changing the polarity of voltage applied, alternating functions (such as arm up and arm down) are obtained. The control module can actuate the left wheel, the right wheel, the arm movement, the wrist movement, and the wrist turn/fingers. Its seven control lines include one for each of the five motions just mentioned plus one each for positive and negative voltage.

As shown in the schematic diagram of Figure 1, the interface is a straightforward design that replaces the control module switches with seven low current relays. These relays are activated by seven transistors which are in turn controlled by seven of the eight available C–64 user port lines. Low current relays insure that the 100 milliamp maximum allowable current drain from the user port is not exceeded.

**Figure 1: Schematic Diagram**




## Construction

Begin construction by fabricating a doubled sided printed circuit board (PCB) using the circuit patterns shown in Figure 2. If you prefer, you can fabricate a single sided PCB by eliminating the top circuit pattern. In this instance, you will have to connect a

jumper wire between hole A and connector pin 2 (+5 volts). While this method does not provide as much rigidity (only half as many connector pins soldered to the PCB), it is somewhat less intimidating. If you opt for the double sided PCB, be sure to insert a short length of wire into hole A and solder it to the pads on either side of the PCB. This replaces the wire jumper used on the single sided version.

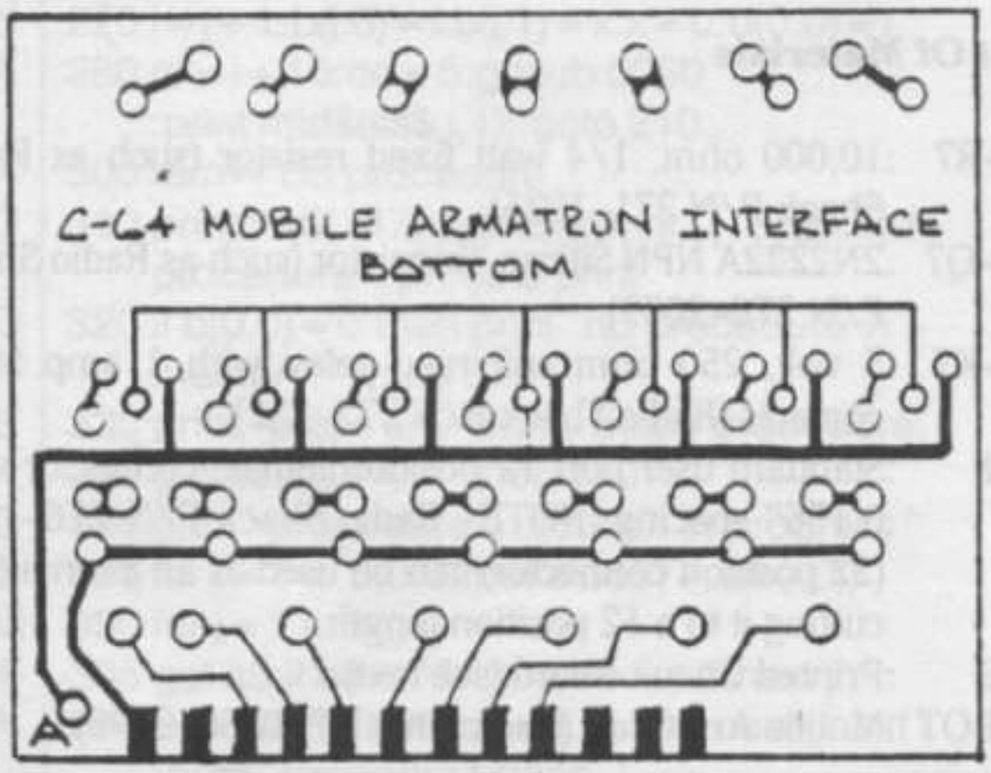
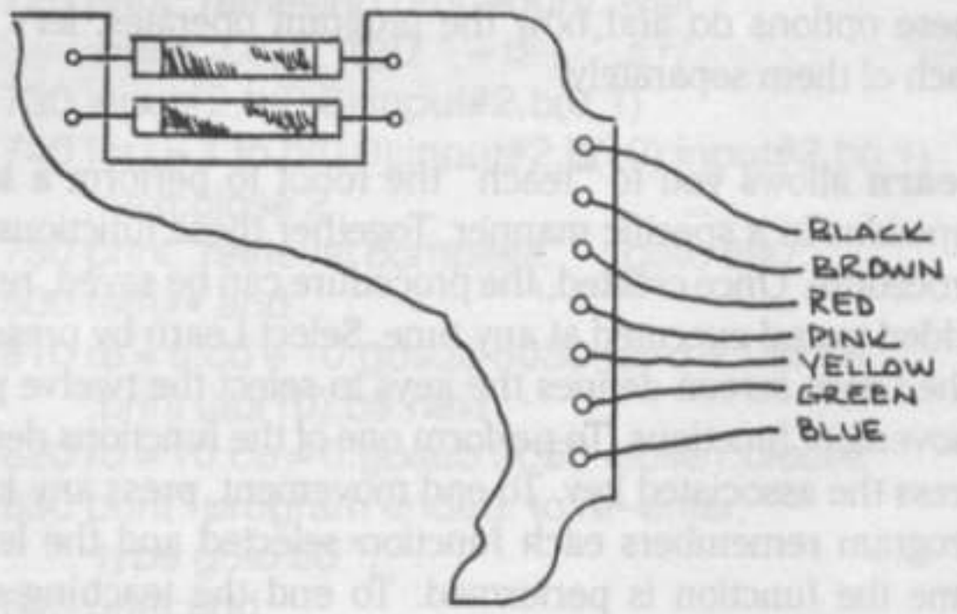


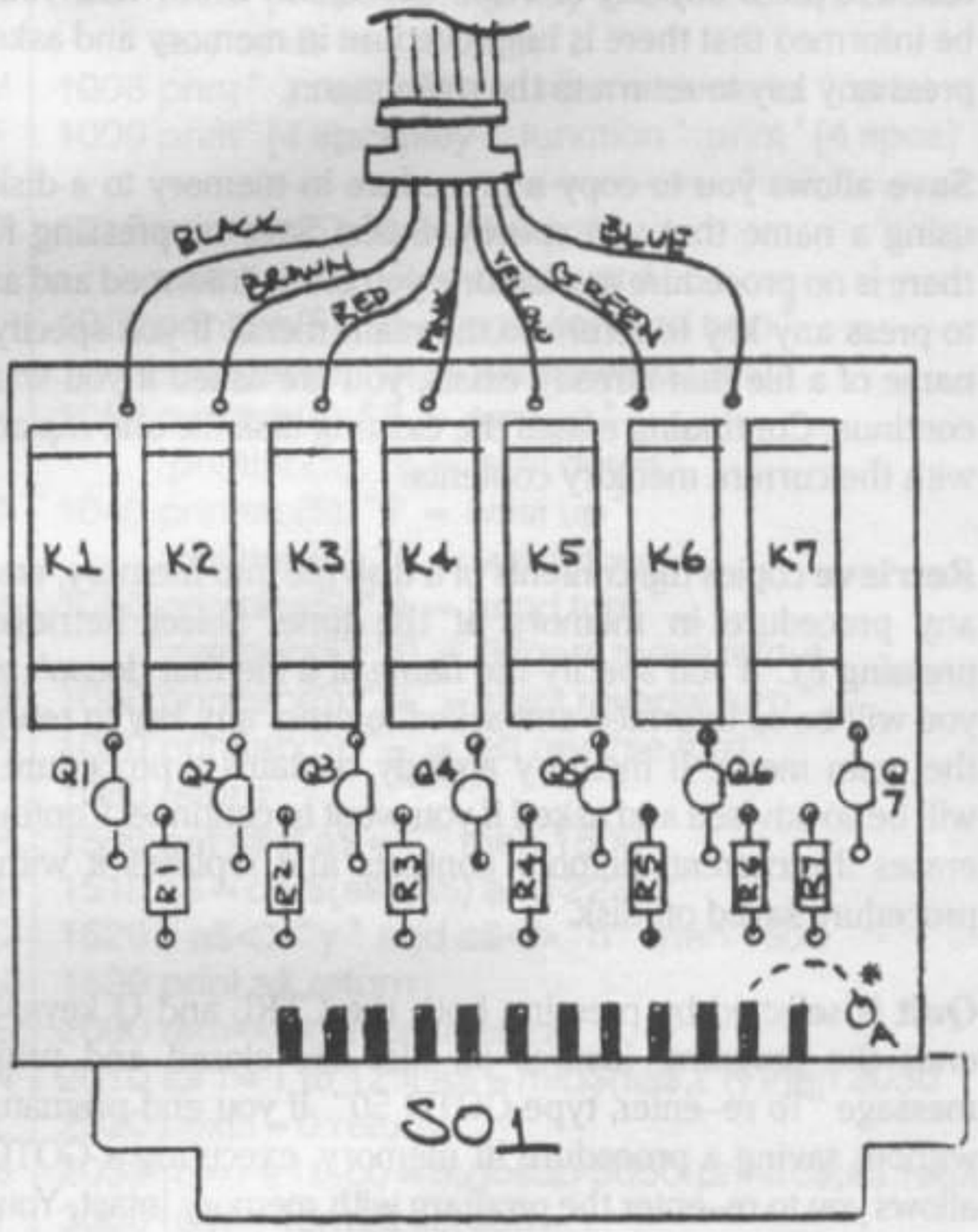

**Figure 2: PC Layout**

Mount the components on the PCB as shown in Figure 3, being sure not to apply excessive heat during the soldering process. When all components have been soldered in place, slide the connector onto the edge of the board so the pins are over the edgeboard pads. Solder each pad to its associated connector pin.

Place the controller module face down and remove the six screws holding it together. Remove the rear case half and note the PCB held in place by a single screw. Remove that screw and the PCB. Turn the PCB over (circuit side) and note the seven wires soldered to the upper edge. Starting from the left they are black, brown, red, pink, yellow, green and blue. Unsolder the wires from the PCB and discard the controller module. Solder these wires to the seven remaining holes as shown in Figure 3. This completes construction.

**Location of wires on Controller PC Board**




* **Double Sided:** Wire through hole soldered to pads on both sides.
**Single Sided:** Wire jumper soldered to pad on other side and Pin 2 (see dotted line).

**Figure 3: Component Placement**

**Use**

Install four D size batteries in the Mobile Armatron. Connect the interface to the user port (rear left of the C–64) such that the component side of the PCB is on top. Power up your C–64 and, if you haven't already, type in the program shown in the Program

Listing and save it using the name ROBOT. When you RUN the program the main menu will appear with the five options Learn, Do, Save, Retrieve and Quit (End). To better understand what these options do and how the program operates, let's discuss each of them separately.

**Learn** allows you to "teach" the robot to perform a series of functions in a specific manner. Together these functions form a procedure. Once created, the procedure can be saved, retrieved, added to and executed at any time. Select Learn by pressing F1. The Learn screen defines the keys to select the twelve possible movement functions. To perform one of the functions described, press the associated key. To end movement, press any key. The program remembers each function selected and the length of time the function is performed. To end the teaching session, press the F1 key to return to the main menu.

**Do** executes a procedure resident in memory. Select Do by pressing F3. If a procedure is resident in memory, you will be asked to press any key to begin execution. Otherwise, you will be informed that there is no procedure in memory and asked to press any key to return to the main menu.

**Save** allows you to copy a procedure in memory to a disk file using a name that you specify. Select Save by pressing F5. If there is no procedure in memory, you are so informed and asked to press any key to return to the main menu. If you specify the name of a file that already exists, you are asked if you want to continue. Continuing erases the existing disk file and replaces it with the current memory contents.

**Retrieve** copies the contents of a disk file into memory, erasing any procedure in memory at the time. Select Retrieve by pressing F7. If you specify the name of a file that doesn't exist, you will be so informed and asked to press any key to return to the main menu. If memory already contains a procedure you will be so advised and asked if you want to continue. Continuing erases the current memory contents and replaces it with the procedure saved on disk.

**Quit** is selected by pressing both the CTRL and Q keys. This ends the program, insures all files are closed and prints a message "To re-enter, type GOTO 50". If you end prematurely without saving a procedure in memory, executing a GOTO 50 allows you to re-enter the program with memory intact. You can then save memory contents to a disk file.

### Experimenting Tips

Each movement you specify is called a step. After loading a procedure from disk, you can add steps to it in the Learn mode. This allows you to build a procedure a part at a time, add to it, test it, and save the revised procedure if it proves acceptable. You can also have standard procedures on file rather than having to build them manually each time.

You should always start a procedure with the robot in the same initial position. To do this, save any procedure in memory and enter the Learn mode. Perform the functions necessary to get the robot to the initial position. Then retrieve the procedure you wish to perform.

Each procedure step is saved as a movement function number and a time count. The movement function is specified by a number between one and twelve (corresponding to the twelve function definitions listed on the Learn screen). The time count is a number that indicates how long the function should be performed. As the robot's batteries drain, the time count may produce slightly different results, since the robot will now move at a slower speed. For precise procedures, then, always use a fresh set of batteries.

### List Of Materials

R1–R7 :10,000 ohm, 1/4 watt fixed resistor (such as Radio Shack P/N 271–1335).
Q1–Q7 :2N2222A NPN Silicon Transistor (such as Radio Shack P/N 276–2009).
K1–K7 :5 volt, 250 ohm coil reed relay with 1 amp SPST contacts (Radio Shack P/N 272–232).
SO1 :Standard user port 12 position female connector with 0.156" spacing. NOTE– Radio Shack P/N 276–1551 (22 position connector) can be used as an alternate by cutting it to a 12 position length.
PCB :Printed Circuit Board (see text).
ROBOT :Mobile Armatron (Radio Shack P/N 60–2396).

MISCELLANEOUS: Solder, short length of wire (see text), four D cell batteries.

NOTE: A disk containing the ROBOT program plus a series of demonstration procedures is available for $6.00 US ($6.36 for New Jersey residents) from B&BTC, RD#1, Box 241H, Tennent Road, Manalapan, NJ, 07726.

### Mobile Armatron Program

```
NL  1 rem ************************************
IC  2 rem **  mobile armatron software       **
CA  3 rem **  name: robot                    **
GK  4 rem **  (c) 1986, jjb                  **
DJ  5 rem **  manalapan, nj  07726           **
JI  6 rem **  v 860928                       **
DM  7 rem ************************************
IF  10 dim a(12),b(250,1):gosub 3000
        :gosub 5000:poke 56579,255:poke 56577,0
HM  20 a(1)=38:a(2)=70:a(3)=36:a(4)=34
        :a(5)=33:a(6)=65
HD  30 a(7)=40:a(8)=72:a(9)=48:a(10)=80
        :a(11)=66:a(12)=68
DA  40 ss$="1234567890+-"
        :rc$=chr$(5)+chr$(18)
CJ  50 gosub 3000:ro=6:co=10:gosub 5050
        :printrc$;"  m a i n   m e n u  "
IJ  60 print:printtab(12);"<f1>: learn":print
        :printtab(12);"<f3>: do"
DA  70 print:printtab(12);"<f5>: save":print
        :printtab(12);"<f7>: retrieve"
KL  80 print:printtab(10);"<ctrl> q: quit (end)
DM  90 ro=19:co=10:gosub 5050:print"which. . . "
HJ  100 get sr$:if sr$="" then 100
ND  110 sr=asc(sr$)-132:if sr=-115 then sr=5
IM  120 if sr<1 or sr>5 then 90
AM  130 on sr gosub 200,300,400,600,800
```

```
NP | 140 goto 50
KL | 200 gosub 1000:j=b(0,0):x=0:rem*** learn
   |     mode ***
GC | 210 get a$:if a$="" then 210
KA | 220 if asc(a$)=133 then return
PM | 230 gosub 2000:if i=0 then poke 56577,0
   |     :goto 210
NJ | 240 poke 56577,a(i)
DD | 250 get a$:if a$="" then x=x+1:goto 250
BN | 260 poke 56577,0
KN | 270 j=j+1:b(j,0)=i:b(j,1)=x:x=0:b(0,0)=j
LA | 280 ro=i+10:co=5:gosub 5050
   |     :print mid$(ss$,i,1);:goto 210
GA | 300 rem** do procedure
PG | 310 printchr$(147):printbl$:print"[13 spcs]do
   |     procedure":printbl$:print
MD | 320 if b(0,0)=0 then print"no procedure in
   |     memory.":goto 375
GE | 330 print"press any key to begin procedure."
PK | 340 get a$:if a$="" then 340
CG | 350 print"procedure execution in progress"
PG | 355 for i=1 to b(0,0):poke 56577,a(b(i,0))
HM | 360 for j=1 to b(i,1)
MF | 365 get a$:if a$="" then x=x+1:next j
JA | 370 poke 56577,0:for j=1 to 500:next j:next i
   |     :print"procedure done."
JL | 375 ro=18:co=0:gosub5050:print"press any key
   |     to return to the menu."
PN | 380 get a$:if a$="" then 380
CK | 390 return
FA | 400 rem** save procedure
OD | 410 printchr$(147):printbl$:print"[13 spcs]save
   |     procedure":printbl$:print
EJ | 420 if b(0,0)=0 then print"no procedure in
   |     memory.":goto 480
JI | 430 input"enter file name to save";f$
FO | 440 open 1,8,15:open 2,8,2,"@0:"+f$+",s,r"
   |     :input#1,e,ed$,tn,bl:close 1:close 2
CG | 450 if e=62 then 500
AB | 460 print"file exists. continue (y/n). . .";
   |     :gosub 1500
JC | 470 if a$="y" then 500
MC | 480 print"abort. press any key. . ."
AF | 490 get a$:if a$="" then 490
LA | 495 return
PD | 500 open 2,8,2,"@0:"+f$+",s,w":print
   |     :print"saving procedure. wait."
AD | 510 for i=0 to b(0,0):print#2,b(i,0):print#2,b(i,1)
   |     :next i:close 2:close 15
OM | 520 print:print"procedure saved. press any key."
   |     :goto 490
LO | 600 rem** retrieve procedure
EC | 610 printchr$(147):printbl$:print"[11 spcs]retrieve
   |      procedure":printbl$:print
IB | 620 if b(0,0)=0 then 650
FL | 630 print"procedure in memory. continue (y/n)?";
   |     :gosub 1500
MA | 640 if a$="n" then print"abort. ";:goto 700
JF | 650 input"enter file name to retrieve";f$
BM | 660 open 1,8,15:open 2,8,2,"@0:"+f$+",s,r"
   |     :input#1,e,ed$,tn,bl:close 1:close 2
GG | 670 if e=0 then 720
BK | 680 if e=62 then print"file doesn't exist. ";
IM | 690 print"press any key."
DB | 700 get a$:if a$="" then 700
CO | 710 return
IG | 720 print"retrieving procedure. wait."
   |     :open 2,8,2,"@0:"+f$+",s,r"
JO | 730 input#2,b(0,0):input#2,b(0,1)
PG | 740 for i=1 to b(0,0):input#2,b(i,0):input#2,b(i,1)
   |     :next:close 2
CB | 750 print"retrieval complete. ";:goto 690
BB | 800 rem** end
AI | 810 ro=5:co=10:gosub 5050:for q=1 to 16
   |     :print tab(10);b$:next
GL | 820 ro=10:co=0:gosub 5050:close1:close2
CM | 830 print"program ended. to re-enter,
   |     type goto 50";
IC | 840 print:end
HK | 1000 rem** learn mode screen
MH | 1005 printchr$(147):printbl$
AH | 1006 print"[3 spcs]mobile armatron robot learn
   |     mode[3 spcs]": printbl$:print
OC | 1007 print"[4 spcs]press key to do function. press
   |     any other key to stop.";
HM | 1008 print"  press <f1> to return to menu.":print
CF | 1009 print"[4 spcs]key   function":print"[4 spcs]
   |     ---  ----------"
MM | 1010 printtab(5);"1 = forward"
   |     :printtab(5);"2 = backward"
IL | 1020 printtab(5);"3 = right forward turn"
   |     :printtab(5);"4 = left forward turn"
MH | 1030 printtab(5);"5 = arm up"
   |     :printtab(5);"6 = arm down"
FD | 1040 printtab(5);"7 = wrist up"
   |     :printtab(5);"8 = wrist down"
AH | 1050 printtab(5);"9 = hand turn"
   |     :printtab(5);"0 = fingers move in/out"
OH | 1060 printtab(5);"+ = right reverse turn"
FP | 1070 printtab(5);"- = left reverse turn"
EF | 1080 return
HP | 1500 get a$:if a$="" then 1500
ID | 1510 a$=chr$(asc(a$) and 223)
FC | 1520 if a$<>"y" and a$<>"n" then 1500
FN | 1530 print a$:return
CE | 2000 rem** position in string
DN | 2010 for i=1 to 12:if a$=mid$(ss$,i,1) then 2030
CL | 2020 next:i=0:return
HB | 2030 ro=i+10:co=5:gosub 5050:printrc$;a$:return
IB | 3000 rem** format screen=
KN | 3010 poke 53280,6:poke 53281,6:printchr$(147)
   |     :b$=chr$(5)+chr$(18)
MM | 3020 bl$=b$+"[39 spcs]":print bl$
DB | 3030 print b$;"  mobile armatron robot controller  "
MC | 3040 printbl$
IA | 3050 b$="[29 spcs]"
AB | 3060 return
OP | 5000 rem* cursor control using plot kernel ($fff0)
NO | 5010 data 162,0,160,0,24,32,240,255,96,999
FK | 5020 a=49300:sc=a
HI | 5030 read b:if b<>999 then poke a,b:a=a+1
   |     :goto 5030
MM | 5040 return
NK | 5050 poke sc+3,col:poke sc+1,row:sys sc
AO | 5060 return
```

# Symlister – A List Formatter for Symass

John A. Spencer
Edwardsville, IL

*. . .Symlister will automatically detect spelling errors in opcodes. .*

Symass is a first class assembler for the C–64 but one feature is noticeably absent. You cannot indent the opcodes to make the labels more easily identified. This is not the "fault" of Symass but rather stems from the fact that the normal Basic operating system is used to manipulate the program. When a line is crunched, the spaces between the line number and what follows are always removed. On listing, a single space is uniformly inserted after each line number to make the program more "readable". But when working with assembly code it is very helpful to have the labels separated to the left of the opcodes. The lack of this formatting can lead, at the least, to eye strain or even worse, to inadvertent duplication of labels. Symlister solves this problem.

This short utility routine, when assembled with Symass 3.13 and enabled, will tidy up your code by automatically indenting each line that begins with a 6510 mnemonic. Because Symlister uses the IQPLOP vector ($306), it also works with listings to a printer in the CMD mode. With Symlister active, enter your assembly code as usual. You need not indent on lines starting with opcodes. When the program is listed, they will be indented. When editing, it is most sensible to align the opcodes appearing after the labels since the indents are fixed for the other lines.

Since Symlister affects only the listing of a program, it does not change the amount of disk space required to store the PRG file.

To conserve programming space, this utility has been designed to hide in the tape buffer. It can be assembled below Symass in memory but when calculating a new starting address, be sure to include an extra 180 bytes or so to allow for the Symass symbol table required during its assembly. Add the following lines to protect the formatter from being overwritten by new Symass symbol tables after it is enabled:

```
1401 lda #<vecset ; protect code
1402 sta $37      ; from symass
1403 lda #>vecset
1404 sta $38
1405 jsr  $a65e   ; reset memory pointers
```

Of course, new enable and disable SYS addresses will have to be calculated for alternate locations. For example, if you load Symass first, it will reside at $952C and Symlister will assemble properly with a starting address of $9400. Then to enable, SYS37888, and disable, SYS37891 (enable+3)

This routine automatically adjusts for variations in the load address of Symass. If you install another utility (eg. C–64 Tiny Aid) before Symass the starting address changes. This adjustment is important because the routine calls two procedures within Symass itself – WORD and FINDOP. A very useful result of this is that Symlister will automatically detect spelling errors in opcodes by not indenting them when you re–list the line. This does not apply to opcodes on lines with labels.

The program is written to automatically vector to whatever was originally in IQPLOP, so if you are using a cartridge that alters this vector it should continue to work normally. However, be careful to disable Symlister with the appropriate SYS before disabling your other utility.

As presented here, Symlister indents eight spaces. This figure works very well with Unassembler files generated from the STP program that converts the SEQ disk files to Symass compatible PRG form. If you favor a different spacing, change the "SPACES" variable before assembling. If Symlister is already active, you can change the spacing by a POKE to location 917 with the desired value.

This program is compatible with Tiny Aid to make it easy to revise the 'ADxxxx' labels from Unassembler. But note that Tiny Aid uses its own display routine so it will not indent the results of FIND or CHANGE operations. Once changes are made, re–list the lines in question to see them formatted. This version of Symlister is not compatible with Verifizer since they both use the tape buffer. Assembly at $C000 (change line 1160 to * = $C000) would be a simple alternative provided you are careful to disable it before Symass does any assembling into this region. In this case enable is SYS49152 and disable SYS49155.

## Symlister Source Code In Symass Format

```
GI 1000 rem save "0:symlister",8
OH 1010 rem * 21 feb 87 - j.a.spencer
CJ 1020 rem * indents all lines starting
AM 1030 rem * with 6510 opcodes
-- 1040 rem * symass must be installed
OI 1050 :
CD 1060 if peek(700)=76 then 1120
CK 1070 :
PE 1080 print "** symass not installed **"
KP 1090 print "load/run symass first"
ME 1100 end
KM 1110 :
DO 1120 print "sys 828 to enable"
EG 1130 print "sys 831 to restore previous list vector"
IO 1140 :
PE 1150 sys700
BM 1160 *=$033c
IA 1170 ;
KK 1180 spaces = $08      ;# spaces to indent
EL 1190 srcad  = $5f      ;code address
NA 1200 ad     = $7a      ;chrgot & symass
GF 1210 iqplop = $0306    ;std list jmp vec
HD 1220 symptr = $02bd    ;symass vector
DG 1230 reglst = $a71a    ;normal list code
FM 1240 outdo  = $ab47    ;print char
KJ 1250 findop = $0000    ;dummy value
EJ 1260 word   = $0000
MG 1270 ;
NG 1280         jmp vecset    ;enable
MO 1290         jmp lisdis    ;disable
KI 1300 ;
AB 1310 ;alter the list vector (iqplop)
MP 1320 vecset  =   *
JN 1330         lda iqplop    ;pick up
DG 1340         sta rgl+1     ;current
AG 1350         lda iqplop+1  ;'list'
BK 1360         sta rgl+2     ;vector
FO 1370         lda #<lvec    ;install new
KD 1380         sta iqplop    ;'list'
KG 1390         lda #>lvec    ;vector
KN 1400         sta iqplop+1
FL 1410         lda symptr    ;calc addr of
LK 1420         clc           ;'findop'
KB 1430         adc #$9d      ;routine
DC 1440         sta opfd+1
KJ 1450         lda symptr+1  ;& insert in
IF 1460         adc #$03      ;code below
DE 1470         sta opfd+2
OD 1480         clc
FA 1490         lda symptr    ;calc addr of
HF 1500         adc #$fa      ;'word'
JC 1510         sta wrd+1     ;routine
GJ 1520         lda symptr+1  ;in symass
DC 1530         adc #$05
LE 1540         sta wrd+2
KP 1550         rts
OI 1560 ;
AG 1570 ;new routine for listing
EI 1580 lvec    cpy #$04      ;check only
JE 1590         beq opsrch    ;1st word
GL 1600 ;
PI 1610 reg     lda (srcad),y ;on line
KM 1620 ;
KH 1630 rgl     jmp reglst
ON 1640 ;
DM 1650 opsrch  sty $49
PM 1660         jsr calcad
MP 1670 ;
GC 1680 wrd     jsr word      ;get len of
MF 1690         jsr calcad    ;word ($59)
KB 1700 ;
OC 1710 opfd    jsr findop
NI 1720         bcc jreg      ;not mnemonic
ID 1730 ;
DB 1740         ldx #spaces
ME 1750 ;
PA 1760 con2    lda #$20      ;insert spaces
DB 1770         jsr outdo
LK 1780         dex
LH 1790         bne con2
OH 1800 ;
BH 1810 jreg    ldy $49       ;restore y reg
PE 1820         bne reg
MJ 1830 ;
HD 1840 ;subroutine to set up search
OD 1850 ;address 'ad' in symass
CM 1860 calcad  =   *
HM 1870         lda $49       ;add .y to
MO 1880         clc           ;current
FP 1890         adc srcad     ;addr
LN 1900         sta ad
JJ 1910         lda #$00
JM 1920         adc srcad+1
LL 1930         sta ad+1
AI 1940         rts
EB 1950 ;
CH 1960 ;sys to disable list formatter
EL 1970 lisdis  =   *
OO 1980         lda rgl+1     ;restore last
OG 1990         sta iqplop    ;vector
GN 2000         lda rgl+2
MD 2010         sta iqplop+1
AN 2020         rts
EG 2030 ;
EN 2040 .end
```

# XREF for the Commodore 64

David Archibald
Flint, Michigan

An indispensible BASIC programming aid

When I'm writing a BASIC program, I find that there are certain programming aids that I use over and over again. The fact is, these programs save me so much time and aggravation that I would find myself hard pressed to get along without them.

One of these indispensable programs is my variable and line number cross reference program, XREF. What is does, for those who are unfamiliar with this type of program, is create a listing of 1) program lines that are referenced by other program lines, and 2) all variables have been used, and on what program lines they appear.

Learning to use Xref is quite simple, since there are only two commands: XREF, and HELP. There are also, only two items of syntax: @ and *. Altogether, there are seven different combinations of these commands and syntax, but I'm sure you won't have any problem remembering them.

But before we get into the explanations of these commands, let's get the basics of running Xref out of the way. All of the commands listed here must be at the beginning of the command line. If they are not, you will get a SYNTAX ERROR. These commands will work only in direct mode; they will not work from within a Basic program. All spaces in the command line will be ignored. The output, which can be sent to the screen or your printer, is listed in numerical or alphabetical order. For this program to do anything, you must of course have a Basic program in memory. To pause the listing, press and hold down the "P" key, until the listing stops. You can restart the listing by pressing any key (just close your eyes, and hit any key that catchs your fancy). You can abort the listing, by pressing RUN/STOP.

And now that you have committed all that to memory, we can start on the commands.

In the following commands, "var-#" can equal any variable name, or line number up to five characters long. Anything over this five character limit is ignored. The symbol <cr> represents pressing the Return key.

XREF @ <cr>

This command sends a complete line number and variable cross reference to your screen.

XREF* @ <cr>

This is the same as the above command, except the output is sent to the printer.

XREF@ var-# <cr>
or XREF*@ var-# <cr>

These commands list all references from the variable or line number onwards. If, for example, you entered the variable "CB$", then the variable CB$ (if it exists) and those alphabetically greater will be listed. You can also use these commands to separately list either the line number or the variable cross reference. In other words, when you enter a line number, then only line number cross references are listed, and the same goes for the variables. Using zero will display all line number cross references, and "A" will display all variables.

XREF var-# <cr>
or XREF* var-# <cr>

With this command, only the single variable or line number that you enter is cross referenced. I like to make use of this command before using a new variable, so I can make sure it hasn't already been used.

XREF <cr>

Each time the command XREF is entered, it will consecutively list a Basic line where the previous "var-#" appears. Lets say for example you enter the command "XREF C$", and you get a listing saying C$ appears on these line numbers: 10,30,100, and 165. If you then enter the command XREF, line 10 will be listed. If you enter it again, then line 30 will be listed. Each time the command XREF is entered, the next Basic line where C$ appears is listed. After the last line is listed (in this case 165), subsequent XREF commands will give you the message "END OF TEXT".

This command is especially handy for editing. Say for example you have ten Basic lines that reference a subroutine at line 1500, and you need to change six of them to call a new subroutine at line 2000. Well, you can either try to remember the six line numbers, and then list them one at a time, or you can simply enter the command XREF 1500, and then follow it with XREF until the six lines pop up. I think I prefer second method, thank you!

HELP <cr>

This is the last command, and the most important. Why is it the most important? Because if you forget everything you have just read, you can enter this command, and get an abbreviated description of these instructions. So if you only remember one command, make it this one.

## Technical Notes

Xref is written completely in machine language, but since It's loaded with a Basic loader, you do not need to know anything about machine language to use it.

Before the Basic loader pokes Xref into memory, it gets the address of the top of Basic memory from locations 55 and 56. It then subtracts 214 (the number of bytes used by Xref) from this address, and Pokes Xref into memory starting from there. What this all means is that Xref is a relocatable program. If for example you already have a machine language program at the top of Basic memory, then the Basic loader will relocate Xref below this other program.

**IMPORTANT NOTE!** Be sure you save the Basic loader before you run it, because it executes a NEW command after it pokes Xref into memory.

Are you wondering how Xref manages to use only 214 bytes of memory? Well, it actually uses over 2000 bytes of memory. So where's the rest? It's hidden under the Basic ROM.

This is made possible because of the Commodore 64's unique memory architecture. What makes it unique is that underneath the ROM for the Basic interpreter and the Kernal is a whole bunch of unused RAM. You can write to this RAM from Basic, but to read it, you really have to use machine language. To read this RAM, you first have to "switch out" the Basic interpreter's ROM. And of course when you switch out the Basic interpreter – which is after all, just a machine language program – it no longer exists in memory, so it is not there to read any memory locations!

Which memory the computer "sees" – the ROM or the RAM – is controlled by the value in zero page memory location one. If bits zero and one in this byte equal zero, then the computer sees RAM at addresses 40960–49151 ($A000–$BFFF), and 57344–65535 ($E000–$FFFF). If these bits equal one, then the computer sees ROM at these addresses.

Xref uses this switching act between RAM and ROM to its advantage, and by doing so saves Basic programming memory. The majority of the Xref program, doesn't care if the Basic interpreter is switched in or out, so most of it is hidden under the ROM. The 214 bytes that are in the Basic programming memory consist mainly of subroutines that switch the ROM back in, and then call other subroutines in the Kernal. When the Kernal subroutine is finished, the Xref subroutine switches the ROM back out, then returns to the part of the main program that called it.

Once Xref is in memory, you don't have to worry that pressing RUN/STOP and RESTORE will kill it, because only turning your computer off will do that.

If you'd rather not type in the program, you can get this issue's Transactor disk, or send $3, a formatted diskette, and a self-addressed, stamped mailer to:

David Archibald
3717 Aldon Lane
Flint, MI 48506

## XREF 64: BASIC Loader

```
GM  10 rem************************************
JL  20 rem**                xref              **
EK  30 rem**    (c) 1985 by david archibald   **
LF  40 rem**         (313) 736-0239           **
OO  50 rem************************************
KF  60 rem
EG  70 rem
OG  80 rem
PN  90 print"[CLR]":print:print:print:print:print:print:print
JF  100 printspc(8)"[RVS]poking xref into memory.":print
BN  110 printspc(6)"please wait approx. two min."
GJ  120 rem
CD  130 rem-- save border color.
KK  140 rem
BK  150 m1=53280:lt=15:p4=1
LJ  160 bc=peek(m1)
IM  170 rem
IC  180 rem-- get top of free memory and
NE  190 rem-- where to put start of prog.
GO  200 rem
LE  210 ea=peek(55)+peek(56)*256-1:sa=ea-214
        :exe=sa
KP  220 rem
HC  230 rem-- check if start of xref is
EB  240 rem-- lower then end of basic prog.
IB  250 rem
OO  260 bm=peek(45)+peek(46)*256+100
EJ  270 if bm>sa then print:print"there isn't enough free
        memory":end
GD  280 rem
MO  290 rem-- convert start of xref into
LD  300 rem-- split decimal.
EF  310 rem
DP  320 th=int(sa/256):tl=sa-th*256
CL  330 for pa=sa to ea
MP  340 read by:tst=by-int(by)
MH  350 rem
BA  360 rem-- randomly change border color.
AJ  370 rem
NN  380 cm=cm+p4:ifcm=20 then p1=rnd(1)*lt+p4
        :pokem1,p1:cm=0
EK  390 rem
FI  400 rem-- if byte followed by .1 then
PC  410 rem-- lower byte of absolute addr.
MA  420 rem-- add lower byte of start to it
LB  430 rem-- so it points to relocated addr
GN  440 rem
LJ  450 if tst>0 and tst<.2 then carry=0:by=by+tl
KO  460 rem
BM  470 rem-- if lower byte is greater than
HE  480 rem-- 255 then a carry is generated
GO  490 rem-- so add 1 to high byte.
CB  500 rem
BH  510 if by>255.5 then by=by-256:carry=1
GC  520 rem
JA  530 rem-- if byte followed by .2 then
IG  540 rem-- high byte of absolute add.
EE  550 rem
OP  560 if tst>.19 then by=by+th+carry:carry=0
HB  570 poke pa,by
```

```
MD 580 next pa
MG 590 rem
GH 600 rem
NM 610 rem-- starting addr ($a000) & ending
MO 620 rem-- addr ($a71e) of main program
EK 630 rem-- under basic's rom.
OJ 640 rem
PC 650 if a=0 then a=1:sa=40960:ea=42781:goto 330
GO 660 pokem1,bc
IK 670 sys exe: new
GA 1000 rem
AB 1010 rem
JJ 1020 rem-- relocatable section of the
LB 1030 rem-- program is not poked under
PI 1040 rem-- basic's rom.
ID 1050 rem
CE 1060 rem
DG 1070 data 32,31.1, 0.2, 76,234,166, 70, 69, 82
AL 1080 data  0, 80, 76, 69,160,  0,132,122,177
NB 1090 data 122,164, 32,166, 33,201, 58,176,  3
CP 1100 data 76,128,  0, 96,120,165,  1, 41,254
AN 1110 data 208,  5, 88,165,  1,  9,  1,133,  1
FJ 1120 data 96, 72, 32,38.1, 0.2,104, 32,210,255
CL 1130 data 76,31.1, 0.2, 32,38.1, 0.2,169,  4, 32
PM 1140 data 195,255, 32,204,255,108,  2,  3, 32
DF 1150 data 38.1, 0.2, 76,167,166, 32,38.1, 0.2, 32
KM 1160 data 228,255, 72, 32,31.1, .2,104, 96,132
OP 1170 data 32,134, 33,166,157, 16,178,166,122
JK 1180 data 208,174,201, 88,240, 10,160,  3,162
MF 1190 data 9.1,201, 72,240,  6,208,160,160,  2
EH 1200 data 162, 6.1,142,126.1, .2, 32,115,  0,217
AN 1210 data 6.1, .2,208,139,136, 16,245,173,126.1
EH 1220 data .2,201, 6.1,240,  6, 32,31.1, .2, 76
OM 1230 data 14,161,169,163,133, 26, 32,115,  0
OA 1240 data 201,  0,208,  6, 32,31.1, .2, 76, 40
CE 1250 data 161, 72,169,  4, 32,195,255,104,162
IC 1260 data  3,160, 36,201, 42,208,  7, 32,115
FD 1270 data  0,162,  4,160, 76,140,166,162, 72
IL 1280 data 169,  4,160,255, 32,186,255,169,  0
KA 1290 data 32,189,255, 32,192,255,162,  4, 32
AH 1300 data 201,255, 32,31.1, 0.2, 76,  0,160
MD 1310 rem
EG 1320 rem -- start of main program
PI 1330 rem -- under basic rom
KF 1340 rem
EJ 1350 data 166, 43,232,232,228, 45,208,  9,166
JP 1360 data  44,228, 46,208,  3, 76, 20,167, 32
IH 1370 data  95,163,104,160,  0,132, 75,230, 76
DD 1380 data 201, 64,208, 11,230, 75,132, 76, 32
EE 1390 data 115,  0,201,  0,240, 11,153,197,163
IH 1400 data 153,203,163,200,192,  5,208,238,140
NL 1410 data 196,163,140,202,163,152,240,  6,169
IA 1420 data  0,133, 75,240, 11,162, 11,189,  9
MA 1430 data 164,157,196,163,202, 16,247,166, 43
DG 1440 data 202,134, 32,165, 44,133, 33,169,  0
KL 1450 data 141, 91,161,230, 30,133,  2,141,193
FB 1460 data 162,169, 29,133,251,169,167,133,252
HE 1470 data 173,197,163,201, 58,176,  8,238, 91
AJ 1480 data 161,169,  0,141,187,160, 32,  4,162
MM 1490 data 166,  2,240,  3, 76, 71,162,162,  0
BC 1500 data 201,143,240, 10,201,131,240,  6,201
HN 1510 data  34,208, 18,162, 34,142,160,160, 32
CL 1520 data   4,162,201,  0,240,223,201,  0,208
NA 1530 data 245,240,217,174, 91,161,240, 40,201
BP 1540 data 138,240, 20,201,137,240, 16,201,141
HO 1550 data 240, 12,201,167,240,  8,162,  0,240
LH 1560 data 189,201, 44,208,180,141,187,160, 32
EP 1570 data   4,162,201, 58,176,170,201, 48,144
ID 1580 data 166,176,  8,201, 65,144,165,201, 91
GL 1590 data 176,161,169,202,133, 25, 32, 75,161
OH 1600 data 240, 37,166, 76,208,148,144,  5,141
NK 1610 data 193,162,176,141,169,196,133, 25, 32
FF 1620 data 145,161,144, 20,240, 18,169,202,133
GM 1630 data  25, 32,185,161,169, 29,133,251,169
GH 1640 data 167,133,252, 32,201,161, 76,123,160
AJ 1650 data 169,164,141, 23,161,160,  0,185, 21
KE 1660 data 164,240, 11, 32,210,255,200,208,245
MK 1670 data 238, 23,161,208,240, 76, 20,167, 32
IO 1680 data  84,163,240, 27, 32,  8,163,240, 14
OM 1690 data 160,  0,177,253,133, 20,200,177,253
AL 1700 data 133, 21, 76, 23,167,169, 29,133,251
PN 1710 data 169,167,133,252, 76,  0,163,165, 32
GK 1720 data 133, 27,165, 33,133, 28,169,  1,133
CD 1730 data  29, 32,  4,162,162,  0,208, 24,201
MK 1740 data  91,176, 32,201, 65,176, 24,201, 36
OL 1750 data 240,  8,201, 37,240,  4,201, 40,208
BH 1760 data   4,230, 29,208, 12,201, 58,176,  8
AA 1770 data 201, 48,144,  4,230, 29,208,213,165
HO 1780 data  30,208, 11, 56,165, 32,233,  1,133
FB 1790 data  32,176,  2,198, 33,173, 91,161,240
DA 1800 data   5, 32,178,161,208, 29,160,  0,177
CE 1810 data  25,197, 29,144,  2,165, 29,170,240
AA 1820 data  10,177, 27,200,209, 25,208,  9,202
DB 1830 data 208,246,160,  0,165, 29,209, 25, 96
PB 1840 data 160,  0,165, 29,145, 25,170,177, 27
LI 1850 data 200,145, 25,202,208,248, 96, 32, 84
LE 1860 data 163,240, 24,160,  0,177,251,197, 20
CJ 1870 data 208, 16,200,177,251,197, 21,208,  9
OD 1880 data 200,169,  1, 24,113,251,145,251, 96
KN 1890 data  24,165,251,105,  3,133,251,165,252
FA 1900 data 105,  0,133,252,160,  0,165, 20,145
CB 1910 data 251,200,165, 21,145,251,200,169,  1
ON 1920 data 145,251, 96,169,  1,141, 35,162,165
OK 1930 data  30,240, 18,198, 30,160,  3,177, 32
NF 1940 data 133, 20,200,177, 32,133, 21,169,  5
BH 1950 data 141, 35,162, 24,165, 32,105,  1,133
BC 1960 data  32,165, 33,105,  0,133, 33,160,  0
GK 1970 data 177, 32,240,  5,201, 32,240,206, 96
AI 1980 data 200,209, 32,240,  3,230, 30, 96,200
IC 1990 data 209, 32,208,248,230,  2, 96, 32, 84
PF 2000 data 163,240,116, 32,230,161,169, 13, 32
KI 2010 data  17,167,174,202,163,160,  1,185,202
NC 2020 data 163, 32, 17,167,200,202,208,246, 32
MB 2030 data  32,163,169, 32,133,253,169,167,133
PM 2040 data 254,160,  0,177,253,141,174,163,200
HB 2050 data 177,253,141,175,163,200,177,253, 72
KL 2060 data  32,108,163,104,141,174,163,201,  1
ML 2070 data 240, 18,238,168,162,169, 47,157,176
AI 2080 data 163,232,169,  0,141,175,163,168, 32
AH 2090 data 120,163,169, 32,157,176,163,232,238
HC 2100 data 168,162,169,  0,201,  0,176, 10,169
PH 2110 data  13, 32, 17,167,160,  1, 32, 32,163
EM 2120 data  32, 51,163, 32, 64,163, 32,  8,163
IJ 2130 data 208,174,169,  0,240, 26,174,202,163
```

```
AI 2140 data 189, 202, 163, 157, 196, 163, 202,  16, 247
ML 2150 data 162,   5, 189,  15, 164, 157, 202, 163, 202
NH 2160 data  16, 247,  76,  79, 160, 165,  75, 240,  22
BN 2170 data 198,  75,  32,  95, 163, 169,  64, 141, 197
AG 2180 data 163, 169,   1, 141, 196, 163, 162,   2,  76
FK 2190 data 208, 162, 104, 104, 169,  29, 133, 253, 169
AM 2200 data 167, 133, 254, 160,  40,  32,  97, 163,  76
JK 2210 data  20, 167,  24, 165, 253, 105,   3, 133, 253
LJ 2220 data 165, 254, 105,   0, 133, 254, 165, 251, 197
BC 2230 data 253, 208,   4, 165, 252, 197, 254,  96, 138
KP 2240 data  72, 169,  32,  32,  17, 167, 232, 200, 192
CE 2250 data  11, 208, 245, 142, 168, 162, 104, 170,  96
CM 2260 data 160,   0, 185, 176, 163,  32,  17, 167, 200
FP 2270 data 202, 208, 246,  96,  32,  26, 167, 201,  80
PO 2280 data 208,   6,  32,  26, 167, 170, 240, 250, 201
EC 2290 data   3, 208,  13,  76, 246, 162, 169,  29, 197
OG 2300 data 251, 208,   4, 169, 167, 197, 252,  96, 160
IO 2310 data   0, 185, 208, 163, 240, 248,  32,  17, 167
DB 2320 data 200, 208, 245, 160,  10, 162,   0, 169,  48
KE 2330 data 153, 175, 163, 136, 208, 248, 142, 153, 163
BN 2340 data  56, 173, 174, 163, 249, 186, 163,  72, 173
EF 2350 data 175, 163, 249, 187, 163, 144,  12, 141, 175
OH 2360 data 163, 104, 141, 174, 163, 254, 176, 163, 176
HF 2370 data 228, 104, 224,   0, 208,   7, 189, 176, 163
LK 2380 data 201,  48, 240,   4, 232, 238, 168, 162, 200
CO 2390 data 200, 192,  10, 208, 206,  96,   0,   0,  48
CM 2400 data  48,  48,  48,  48,  48,  48,  48,  48,  48
PK 2410 data  16,  39, 232,   3, 100,   0,  10,   0,   1
DB 2420 data   0,  78,  85,  77,  66,  69,  82,  78,  85
EM 2430 data  77,  66,  69,  82,  13,  13,  18,  86,  65
GM 2440 data  82,  47,  78,  85,  77,  32,  45,  32,  65
PM 2450 data  80,  80,  69,  65,  82,  83,  32,  79,  78
DP 2460 data  32,  76,  73,  78,  69,  32,  78,  85,  77
ON 2470 data  66,  69,  82,  40,  83,  41,  13,   0,  13
PN 2480 data  13,  18,  69,  78,  68,  32,  79,  70,  32
IG 2490 data  84,  69,  88,  84,  13,  13,   0,   1,  48
FO 2500 data  32,  32,  32,  32,   5,  90,  90,  90,  90
AO 2510 data  90, 147,  32,  32,  32,  32,  32,  32,  32
IL 2520 data  32,  32,  32,  32,  32,  32,  32,  18,  88
KA 2530 data  82,  69,  70,  54,  52,  32,  72,  69,  76
CA 2540 data  80,  13,  13,  34,  88,  34,  32,  67,  65
EE 2550 data  78,  32,  69,  81,  85,  65,  76,  32,  65
BF 2560 data  32,  86,  65,  82,  73,  65,  66,  76,  69
LC 2570 data  32,  85,  80,  32,  84,  79,  32,  70,  73
NH 2580 data  86,  69,  13,  76,  69,  84,  84,  69,  82
FG 2590 data  83,  32,  76,  79,  78,  71,  44,  32,  79
FH 2600 data  82,  32,  65,  32,  78,  85,  77,  66,  69
CJ 2610 data  82,  32,  66,  69,  84,  87,  69,  69,  78
HE 2620 data  13,  48,  45,  54,  53,  53,  51,  53,  46
LG 2630 data  13,  13,  18,  34,  88,  82,  69,  70,  64
LH 2640 data  32,  60,  67,  82,  62,  34, 146,  32,  58
BJ 2650 data  80,  82,  73,  78,  84,  83,  32,  65,  76
OK 2660 data  76,  32,  82,  69,  70,  69,  82,  69,  78
JK 2670 data  67,  69,  83,  46,  13,  18,  34,  88,  82
DJ 2680 data  69,  70,  42,  64,  32,  60,  67,  82,  62
OO 2690 data  34, 146,  32,  58,  80,  82,  73,  78,  84
PK 2700 data  83,  32,  65,  76,  76,  32,  82,  69,  70
NP 2710 data  69,  82,  69,  78,  67,  69,  83,  32,  84
CP 2720 data  79,  13,  89,  79,  85,  82,  32,  80,  82
OM 2730 data  73,  78,  84,  69,  82,  46,  13,  18,  34
CO 2740 data  88,  82,  69,  70,  64,  32,  88,  32,  60
IA 2750 data  67,  82,  62,  34, 146,  32,  58,  80,  82
KP 2760 data  73,  78,  84,  83,  32,  65,  76,  76,  32
KD 2770 data  82,  69,  70,  69,  82,  69,  78,  67,  69
CA 2780 data  83,  44,  13,  70,  82,  79,  77,  32,  34
NB 2790 data  88,  34,  32,  79,  78,  46,  13,  18,  34
EC 2800 data  88,  82,  69,  70,  42,  64,  32,  88,  32
AF 2810 data  60,  67,  82,  62,  34, 146,  32,  58,  83
GE 2820 data  65,  77,  69,  32,  65,  83,  32,  65,  66
BF 2830 data  79,  86,  69,  44,  32,  66,  85,  84,  32
NG 2840 data  84,  72,  69,  13,  79,  85,  84,  80,  85
KF 2850 data  84,  32,  73,  83,  32,  83,  69,  78,  84
AH 2860 data  32,  84,  79,  32,  89,  79,  85,  82,  32
LF 2870 data  80,  82,  73,  78,  84,  69,  82,  46,  13
BI 2880 data  18,  34,  88,  82,  69,  70,  32,  88,  34
HG 2890 data  32,  79,  82,  32,  34,  88,  82,  69,  70
II 2900 data  42,  32,  88,  34, 146,  32,  58,  80,  82
KJ 2910 data  73,  78,  84,  83,  32,  84,  79,  32,  84
OM 2920 data  72,  69,  13,  83,  67,  82,  69,  69,  78
GL 2930 data  32,  79,  82,  32,  89,  79,  85,  82,  32
AL 2940 data  80,  82,  73,  78,  84,  69,  82,  32,  65
GP 2950 data  76,  76,  32,  82,  69,  70,  69,  82,  69
CN 2960 data  78,  67,  69,  83,  13,  84,  79,  32,  34
GO 2970 data  88,  34,  46,  13,  18,  34,  88,  82,  69
LL 2980 data  70,  32,  60,  67,  82,  62,  34, 146,  32
NC 2990 data  58,  67,  79,  78,  83,  69,  67,  85,  84
JB 3000 data  73,  86,  69,  76,  89,  32,  76,  73,  83
MB 3010 data  84,  83,  32,  79,  78,  69,  13,  66,  65
HA 3020 data  83,  73,  67,  32,  76,  73,  78,  69,  44
EB 3030 data  32,  87,  72,  69,  82,  69,  32,  34,  88
AB 3040 data  34,  32,  65,  80,  80,  69,  65,  82,  83
PA 3050 data  46,  13,  13,  80,  82,  69,  83,  83,  32
HA 3060 data  84,  72,  69,  32,  34,  80,  34,  32,  75
GG 3070 data  69,  89,  32,  84,  79,  32,  80,  65,  85
CF 3080 data  83,  69,  32,  84,  72,  69,  13,  76,  73
PE 3090 data  83,  84,  73,  78,  71,  44,  32,  65,  78
KI 3100 data  68,  32,  65,  78,  89,  32,  75,  69,  89
DH 3110 data  32,  84,  79,  32,  67,  79,  78,  84,  73
IG 3120 data  78,  85,  69,  46,  13,  80,  82,  69,  83
OJ 3130 data  83,  32,  82,  85,  78,  47,  83,  84,  79
MI 3140 data  80,  44,  32,  84,  79,  32,  67,  65,  78
JJ 3150 data  67,  69,  76,  32,  84,  72,  69,  32,  76
DH 3160 data  73,  83,  84,  73,  78,  71,  46,  13,  13
CB 3170 data   0, 147,  13,  32,  32,  32,  32,  32,  32
HC 3180 data  32,  32,  32,  32,  32,  32,  32,  32,  32
HJ 3190 data  32,  18,  88,  82,  69,  70,  54,  52,  13
ID 3200 data  13,  13,  32,  32,  32,  32,  32,  32,  32
JL 3210 data  32,  65,  32,  86,  65,  82,  73,  65,  66
BO 3220 data  76,  69,  32,  38,  32,  76,  73,  78,  69
FM 3230 data  32,  78,  85,  77,  66,  69,  82,  13,  32
DI 3240 data  32,  32,  32,  32,  32,  32,  32,  67,  82
FP 3250 data  79,  83,  83,  32,  82,  69,  70,  69,  82
OP 3260 data  69,  78,  67,  69,  32,  80,  82,  79,  71
EK 3270 data  82,  65,  77,  13,  13,  32,  32,  32,  32
IN 3280 data  32,  32,  18,  40,  67,  41,  32,  49,  57
KC 3290 data  56,  53,  32,  66,  89,  32,  68,  65,  86
IC 3300 data  73,  68,  32,  65,  82,  67,  72,  73,  66
GG 3310 data  65,  76,  68,  13,  13,   0, 162, 6.1, 160
OA 3320 data 0.2, 134,  55, 132,  56, 169,  76, 162,89.1
ON 3330 data 160, 0.2, 133, 124, 134, 125, 132, 126, 160
MP 3340 data   0, 185, 103, 166, 240,   9,  32,  17, 167
KP 3350 data 200, 208, 245,  76,31.1, 0.2,  76,38.1, 0.2
MB 3360 data  76,46.1, 0.2,  76,57.1, 0.2,  76,71.1, 0.2
PI 3370 data  76,77.1, 0.2,   0
```

# Beyond COMPARE

**David Lathrop**
**Fremont, California**

*Find the differences between two files of any kind –*
*BASIC, assembler, text, database records, or hexadecimal!*

Utilities are usually pretty dull subjects. So a file compare program probably will not generate a lot of excitement –– until you need it. When you do need one, you may discover that there are not many of them available, and they do not handle inserted or deleted records very well. Furthermore, if you compare BASIC or machine language program files, these other compare programs show file differences in binary or ASCII data as they are stored on disk, not as BASIC statements or ML instructions.

COMPARE is a file comparison utility that does not suffer from these shortcomings. It recognizes inserted and deleted lines and makes adjustments before continuing with the comparison. If the files are BASIC programs, COMPARE will list the program files as the LIST command does (that is, the line numbers and tokens will be interpreted). If the files are 6502 machine language programs, they can be disassembled. If the files are text or data, they may be shown as either ASCII text or hexadecimal data.

A simple BASIC program is shown in Figure 1. A modified version of it is shown in Figure 2. Can you easily see the differences? A report created by COMPARE is shown in Figure 3, and this report clearly shows how the two programs differ.

A nice fringe benefit of COMPARE is the ability to list a file in a variety of formats. To do this, the file is compared with "nothing". All the differences between the file and "nothing" are reported, and these differences represent the entire contents of the file. We'll discuss how to do this shortly.

**Program Preparation**

First enter and save Program 1. This BASIC program, named COMPARE, is the main program which loads ML routines and prompts for information such as file names and report formats. It then calls the ML routines to do the comparison.

Next enter, save and run Program 2. This BASIC program creates an ML program named CMP.ML.6800. This ML program reads the files, compares them, and reports the differences.

**How to use COMPARE**

After saving the BASIC and ML programs on disk, load and run COMPARE. After loading the ML routines, COMPARE will prompt you for the following:

1. The address of the drive you wish to use – this is usually 8 or 9.
2. The name of the OLD file – COMPARE assumes that one of the two files pre–dates the other, so the earlier one is called the OLD file. If you are in doubt about the file's name, you can list the diskette directory by entering **$** as the file name.
3. The name of the NEW file – For comparisons, this is the name of the second file. But if you want to list the OLD file, enter an asterisk (*) as the NEW file name.
4. The destination of the comparison report – Choose **s** for screen, or **p** for printer.
5. The report format – If the OLD file is a program file, you may choose **a** (ASCII text), **b** (BASIC list), **d** (disassembled 6502 list), or **x** (hexadecimal data) formats.

If the OLD file is a sequential, user or relative file, you may choose **a** (ASCII text) or **x** (hexadecimal data) format.

The choice of format tells COMPARE how to process the file's records, and how to prepare the display of non–comparing records. If you request either a BASIC or 6502 machine language report, that is sufficient information for the compare to begin. However if you request either ASCII text or hexadecimal data formats, you must tell COMPARE what kind of records these files have.

*If you don't know what the internal structure of the file is, COMPARE can help you by listing the files for inspection. Pick one of the files for the OLD file name, specify * for the NEW file name, select report format* **x** *for hexadecimal, choose* **f** *for fixed record length, and specify a record length of 127 bytes. With a little investigation, you should be able to determine how records are defined in the file.*

COMPARE can work with fixed (**f**) or variable (**v**) length records. If the record length is fixed, COMPARE will ask you what that length is. But if the length is variable, COMPARE will ask how it can determine the end of each record. Three possibilities are allowed:

1. The length of the record is contained in the data. For example, suppose the first byte indicates how long the record is. In this case you would choose option **l** and indicate that the position of the length value is one.
2. The end of the record is marked by a special character. For example, many text files terminate a line or paragraph with a carriage return. In this case you would choose option **c** and indicate that the decimal value of the special character is 13.
3. The end of the record is marked by special bits, the value of which is called a mask. To use this option, you need to understand how eight bit bytes can be expressed as decimal values from 0 to 255. For example, many text files mark the end of a record by setting the most significant bit. In this case you would choose option **m** and indicate that the mask is 128.

For option **c** or **m**, you will be asked for a minimum record length. This is because it is possible that the special character or mask may appear in the first few bytes of a record. For example, a BASIC program file has a minimum record length of four because a zero marks the end of a statement, but a zero may also appear in the first

four bytes as either part of the line number or the line link. This prompt allows you to tell COMPARE to ignore any occurrence of the special character or mask at the start of a record. Specify zero if this is not the case.

At the completion of this step, you will have described to COMPARE the nature of the files to compare and the format of the comparison report you wish to see. Now the comparison will start. Figure 3 shows an example of a comparison between two BASIC program files. Figures 4 and 5 show a 6502 disassembly and a hexadecimal list of another program. Report types a, d, and x show the ASCII representation of data, but all Commodore graphics characters, cursor movement and color characters are shown as periods.

When the comparison begins, a report heading is printed which shows the names of the two files. Following the header, each mismatch between files is grouped into blocks. The records in the OLD file which do not compare are listed first followed by a dashed line (-------). The corresponding part of the NEW file is then listed followed by a line of equal signs (= = = = = = = =). You will note that the last line printed for both files in each block of mismatched records (except possibly the very last block) are identical. This is the point where COMPARE has determined that the files are the same, has realigned them and continued the comparison. A count of the mismatched blocks is shown at the end of the report.

While the comparison is taking place, you may press any of the following keys:

**CTRL** slows the scrolling speed if the comparison report is displayed on your monitor screen.

**F3** terminates the comparison immediately.

**F7** pauses the comparison. This is useful if the report is displayed on the screen and you wish to study it before letting it scroll by. Press any other key to resume.

### Technical Notes

The BASIC and ML programs communicate with each other via variables. Variables associated with the OLD file begin with an "o", and variables associated with the NEW file begin with an "n". The variables are defined before the ML program is called, and are used as follows:

General variables

dn% – disk unit address (input to ML)
pn% – printer or screen address (to ML)
pc% – printer secondary address (to ML)
mm% – mis–match count (output from ML)
rc% – return code (from ML)

OLD file

oo$ – file name padded with shifted spaces (to ML)
ov$ – report format a,b,d,x (to ML)
ot$ – record type f,v,l,c,m (to ML)
oz$ – rec. length, length position, character or mask (to ML)
os$ – minimum record length (to ML)
of$ – 4K buffer RAM page nr (to ML)
ob% – file starting track & sector (to/from ML)
oy% – file type 1=seq,2=prg,3=usr,4=rel (from ML)

NEW file

nn$ – file name padded with shifted spaces
n?? – see OLD file variables above

The COMPARE ML program resides between $6800 and $8000, so it should be able to co–exist peacefully with expansion cartridges and most other ML programs. Page zero is used heavily, but it is saved and subsequently restored before the ML ends. Two 4K file buffers fit between $4000 and $6000, so the BASIC program lowers MEMTOP accordingly (see line 10). If you wish to increase the RAM available to BASIC, you can move the buffers up under BASIC ROM between $A000 and $C000 by setting of$=chr$(160): nf$=chr$(176).

The BASIC program may be modified to suit your tastes if you take care to define the variables shown above before you SYS to any of the ML routines. If you wish to make modifications, the following will be of interest:

If pn%=3, a 40 column report is created to fit on the screen, otherwise an 80 column report is produced. If you have an 80 column cartridge, you can show 80 columns on the screen by opening a file to the screen such as pn%=5:open5,3,0.

You may also write a report to disk by opening a file such as pn%=2:open2,8,2,"comp.report,s,w".

If you wish to display or compare files which do not have entries in the diskette directory, you may do so by setting ob% and nb% equal to the files' beginning track and sector. For example, to list a diskette BAM and directory blocks, set nb%=0: ob%=18*256 before the sys26624.

### Beyond Compare

As good as COMPARE is, it is not foolproof. If two files have a number of identical records in them such as blank lines in a text file or REM statements without any text in a BASIC program, COMPARE can be fooled into thinking it has found a point where two dissimilar files are the same and a block of non–comparing records may be printed prematurely. This is usually not a serious problem, but if these "trivial" records disrupt the comparison, you may wish to make copies of the two files with these records deleted.

All of the records for each file which do not compare are held in RAM until they are printed. If there is a large block of records in either file which does not correspond to any records in the other file, COMPARE may run out of space to hold the records. When this happens, a warning message will be printed, all records currently in memory will be printed and the comparison will continue from that point. This event may cause COMPARE to subsequently report differences between the files when in fact they do not exist.

After you have used COMPARE, you may find it very valuable as a file listing utility. You can use it to examine database files and see how they are structured. It is very helpful in listing BASIC programs with REM statements which confuse the LIST command. Machine

language programs may be disassembled and listed directly from disk. But COMPARE's main function is to compare two files. When you need to do that, there is little else around that works as well. In those situations, this little utility is beyond compare.

**Figure 1: UNNEW.BASIC Program File Listing**

```
100 poke53281,0:print" ..sys525 "
110 i=525
120 reada:ifa=256then130
130 pokei,a:i=i+1:goto120
140 poke43,525 and 255:poke44,2
150 poke45,578 and 255:poke46,2
160 print" .":save"0:unnew",8
170 rem for tape use save"unnew",1,1
180 data 160,   3, 200, 177,  43, 208, 251
190 data 200, 200, 152, 160,   0, 145,  43
200 data 165,  44, 200, 145,  43, 133,  60
210 data 160,   0, 132,  59, 162,   0, 200
220 data 208,   2, 230,  60, 177,  59, 208
230 data 245, 232, 224,   3, 208, 242, 200
240 data 208,   2, 230,  60, 132,  45, 164
250 data  60, 132,  46,  96, 256
```

**Figure 2: UNNEW.BASIC2 Program File Listing**

```
110 i=525
120 reada:ifa=256then130
130 pokei,a:i=i+1:goto120
140 poke43,525 and 255:poke44,2
150 poke45,578 and 255:poke46,2
160 print" .":save"0:unnew",8
170 data 160,   3, 200, 177,  43, 208, 215
180 data 200, 200, 152, 160,   0, 145,  43
190 data 165,  44, 200, 145,  43, 133, 60
200 data 160,   0, 132,  59, 162,   0, 200
210 data 208,   2, 230,  60, 177,  59, 208
220 data 245, 232, 224,   3, 208, 242, 200
230 data 208,   2, 230,  60, 132,  45, 164
240 data  60, 132,  46,  96, 256
```

**Figure 3: BASIC Program Files Comparison Report**

```
**********************************************
*        File Comparison Utility  V3.4       *
**********************************************
OLD Name: unnew.basic
NEW Name: unnew.basic2

100 poke53281,0:print" ..sys525 "
110 i=525
----------------------------------------
110 i=525
========================================
170 rem for tape use save"unnew",1,1
180 data 160,   3, 200, 177,  43, 208, 251
190 data 200, 200, 152, 160,   0, 145,  43
----------------------------------------
170 data 160,   3, 200, 177,  43, 208, 215
180 data 200, 200, 152, 160,   0, 145,  43
========================================
Compare finished. 2 mismatches found.
```

**Figure 4: Machine Language Program Disassembly**

(Editor's Note: COMPARE prints a character representation of the bytes in the column down the right. Some of the characters are graphics that we can't typeset. However, they aren't critical for the example reports, so we've replaced them with a § )

```
**********************************************
*        File Comparison Utility  V3.4       *
**********************************************
OLD Name: unnew
NEW Name: *

020d-  a0 03     ldy  #$03     <. .>
020f-  c8        iny           <H>
0210-  b1 2b     lda  ($2b),y  <§ +>
0212-  d0 fb     bne  $020f    <P§>
0214-  c8        iny           <H>
0215-  c8        iny           <H>
0216-  98        tya           <.>
0217-  a0 00     ldy  #$00     <. .>
0219-  91 2b     sta  ($2b),y  <. +>
021b-  a5 2c     lda  $2c      <§ ,>
021d-  c8        iny           <H>
021e-  91 2b     sta  ($2b),y  <. +>
0220-  85 3c     sta  $3c      <.<>
0222-  a0 00     ldy  #$00     <. .>
0224-  84 3b     sty  $3b      <. ;>
0226-  a2 00     ldx  #$00     <§ .>
0228-  c8        iny           <H>
0229-  d0 02     bne  $022d    <P .>
022b-  e6 3c     inc  $3c      <§<>
022d-  b1 3b     lda  ($3b),y  <§ ;>
022f-  d0 f5     bne  $0226    <P§>
0231-  e8        inx           <§>
0232-  e0 03     cpx  #$03     <  .>
0234-  d0 f2     bne  $0228    <P§>
0236-  c8        iny           <H>
0237-  d0 02     bne  $023b    <P .>
0239-  e6 3c     inc  $3c      <§<>
023b-  84 2d     sty  $2d      <.->
023d-  a4 3c     ldy  $3c      <§<
023f-  84 2e     sty  $2e      <. .>
0241-  60        rts           <  >
```

**Figure 5: Hexadecimal Listing**

```
**********************************************
*        File Comparison Utility  V3.4       *
**********************************************
OLD Name: unnew
NEW Name: *

0000   0d02 a003 c8b1 2bd0   <....H§ +P>
       fbc8 c898 a000 912b   <§HH.... +>
       a52c c891 2b85 3ca0   <§ ,H. +.<.>
       0084 3ba2 00c8 d002   <.. ;§.HP.>

0020   e63c b13b d0f5 e8e0   <§<§ ;P§§  >
       03d0 f2c8 d002 e63c   <.P§HP.§<>
       842d a43c 842e 60     <.-§<.. >
```

## COMPARE: BASIC Portion

```
CN | 10 poke56,64:clr:co=06:cr=02:cb=1:poke646,co
   |    :poke53280,cr:poke53281,cb
GI | 20 if peek(26636)=51 and peek(26638)=52
   |    then100
DL | 30 gosub940:print" sr STANDBY - Loading
   |    ML Subroutines.          "
IC | 40 dn%=peek(186):load"cmp.ml.6800",dn%,1
IO | 100 hd$="*File Comparison Utility  V3.4*"
IO | 110 bx$="[34*'s]"
EL | 120 rem of$=chr$(160):nf$=chr$(176)
OG | 130 mm%=0:rc%=0:cr$=chr$(13):ze$=chr$(0)
   |    :fori=1to16:sp$=sp$+chr$(160):next
HK | 140 ov$="":ot$="":oz$="":os$=""
   |    :ob%=0:oy%=0
BK | 150 nv$="":nt$="":nz$="":ns$=""
   |    :nb%=0:ny%=0
EB | 160 :
PG | 170 gosub940
GI | 180 dn%=0:input"Disk unit nr (8-11)";dn%
   |    :if dn%=0 then end
FD | 190 if dn%<8 or dn%>11 then print"Unit
   |    nr must be 8-11":goto180
JH | 200 gosub940:ol$="":input"OLD file name
   |    or r $ R";ol$
JA | 210 if ol$="$" then gosub970:goto200
DP | 220 if ol$="" then170
HA | 230 nu$="":input"NEW file name or r * R";nu$
MI | 240 if nu$="$" then gosub970:gosub940:goto200
PB | 250 if nu$="" then200
PC | 260 pd$="":input"Report to r s Rcreen
   |    or r p Rrinter";pd$
OP | 270 if pd$="" then230
DL | 280 if pd$="s" then pn%=3:pc%=0:goto310
LL | 290 if pd$="p" then pn%=4:pc%=7:goto310
MN | 300 goto260
KK | 310 :
JF | 320 :rem--- get file type, trk, sector
BK | 330 oo$=left$(ol$+sp$,16)
LK | 340 nn$=left$(nu$+sp$,16)
PB | 350 sys26627:if nu$="*" then ny%=1:nb%=0
IH | 360 if rc%<128 then400
KF | 370 if rc%<255 then a$="failed.":goto390
FJ | 380 a$="cancelled."
GN | 390 print"Directory search ";a$:goto440
EL | 400 if oy%<>0 and ny%<>0 then480
KM | 410 if oy%=0 then print"OLD file not found."
FM | 420 if ny%=0 then print"NEW file not found."
DN | 430 print"Enter r $R to list the disk directory."
JC | 440 print"Press any key to continue"
AC | 450 get a$:if a$="" then450
EG | 460 goto200
KE | 470 :
AP | 480 :rem--- get report and record type
HL | 490 os$=ze$:print"Choose r a Rscii";
DA | 500 if oy%=2 then print",r b Rasic,
   |       r d Risassembly";
JJ | 510 ov$="":input",he r x R";ov$
JI | 520 if ov$="a" or ov$="x" then580
KL | 530 if oy%=2 and ov$="b" then ot$="c"
   |    :oz$=chr$(0):goto820
BN | 540 if oy%=2 and ov$="d" then ot$="f"
   |    :oz$=chr$(1):goto820
AF | 550 if ov$="" then goto820
CP | 560 goto490
PF | 570 :rem--- record specification
JE | 580 ot$="":input"Is record length r f Rixed
   |       or r v Rariable";ot$
FL | 590 if ot$="f" then630
LA | 600 if ot$="v" then660
FK | 610 if ot$="" then490
NC | 620 goto580
NK | 630 x=0:input"Specify record length (1-255)";x
   |    :if x<1 or x>255 then580
OH | 640 oz$=chr$(x):goto820
IE | 650 :rem--- variable In record
AN | 660 print"Choose end of record indicator. . ."
LE | 670 ot$="":input"      r c Rharacter, r l Rength,
   |       or r m Rask";ot$
EJ | 680 if ot$="c" then a$="character":goto730
CF | 690 if ot$="m" then a$="mask":goto730
GF | 700 if ot$="l" then790
IA | 710 if ot$="" then580
MI | 720 goto660
GK | 730 x$="":print"Enter value of ";a$;" (0-255)";
   |    :input x$:if x$="" then660
KC | 740 x=val(x$):if x<0 or x>255 then730
LG | 750 oz$=chr$(x)
CL | 760 x$="":input"Enter minimum record length
   |       (0-255)";x$:if x$="" then730
OE | 770 x=val(x$):if x<0 or x>255 then760
FP | 780 os$=chr$(x):goto820
IN | 790 x$="":input"Specify the location (1-255)";x$
   |    :if x$="" then760
DH | 800 x=val(x$):if x<1 or x>255 then790
HK | 810 oz$=chr$(x)
CH | 820 if ov$="" then260
GC | 830 nv$=ov$:nt$=ot$:nz$=oz$:ns$=os$
ML | 840 :
BD | 850 :rem--- produce comparison report
BE | 860 a$=chr$(147):b$="[3 spcs]":if pn%<>3
   |       then a$=chr$(12):b$="[16 spcs]"
PA | 870 open pn%,pn%,pc%
OL | 880 print#pn%,a$;b$;bx$;cr$;b$;hd$;cr$;b$;bx$;cr$
GH | 890 print#pn%,b$;"OLD Name:  ";ol$
EM | 900 print#pn%,b$;"NEW Name:  ";nu$;cr$
CJ | 910 sys26624:gosub990:goto200
MA | 920 :
ML | 930 :--- display screen header
LK | 940 print" S "chr$(14)tab(3)hd$cr$cr$:return
KC | 950 :
LO | 960 :--- list directory
LA | 970 pn%=3:nu$="*":open pn%,pn%,pc%
   |    :sys26630
IE | 980 :
JL | 990 a$="Compare ":if nu$="*" then
   |       a$="Listing "
KC | 1000 b$="finished.":if rc%>99 then b$="failed."
   |       :if rc%=255 then b$="cancelled."
LK | 1010 if nu$<>"*" then b$=b$+str$(mm%)
   |       +" mismatches."
EN | 1020 print#pn%,cr$cr$a$b$:print#pn%:close pn%
PE | 1030 if pn%<>3 then1060
BN | 1040 print"Press any key to continue."
DI | 1050 get a$:if a$="" or a$=chr$(136) then1050
AE | 1060 return
```

## COMPARE: Creates Machine Language file

```
BH  10 rem* data loader for "compare" *
EM  20 for i=26624 to 31790:read a
JG  30 cs=cs+a: next i
CO  40 if cs<>523626 then print "!data error!": end
NH  50 open 1,8,1,"cmp.ml.6800,p,w"
GJ  55 print#1,chr$(0)chr$(104);
EF  60 restore
LG  70 for i=1 to 5167
NM  80 read a: print#1,chr$(a);: next i
FA  90 close 1: print "ok, ml file has been
        written.": end
IN  100 :
PD  1000 data  76,  38, 105,  76,  21, 106,  76,  92
MC  1010 data 107,  99, 109, 112,  51,  46,  52,  76
FM  1020 data 151, 108,  76, 194, 109,  76,  19, 110
OC  1030 data  76,  79, 110,  76, 111, 110,  76, 151
OC  1040 data 110,  76, 230, 110,  76,  11, 112,  76
GF  1050 data 245, 112,  76,  48, 113,  76,  83, 114
MK  1060 data  76, 101, 114,  76, 110, 115,  76, 209
JP  1070 data 116,  76, 218, 117,  76, 246, 117,  76
KN  1080 data  61, 118,  76, 128, 118,  76, 252, 118
NG  1090 data  76,  50, 119,  76,  86, 119,  76, 103
NC  1100 data 119,  76, 129, 119,  76, 165, 119,  76
CI  1110 data  26, 120,   0,   0,   0,   8,  15,   7
GI  1120 data   3,  85,  49,  58,  55,  44,  48,  44
JB  1130 data  84,  84,  44,  83,  83,   0, 102, 105
JA  1140 data 108, 101, 110,  97, 109, 101,  46,  46
HN  1150 data  46,  46,  46,  46,  46,  46, 119, 114
AI  1160 data 107,  50,   0,  64,   0,  80,  15,   0
AG  1170 data   0,   0,   0,   0,   0,   0,   0,   0
KG  1180 data   0,   0,   0,   0,   0,   0,   0,   0
EH  1190 data   0,   0,   0,   0,   0,   0,   0,   0
OH  1200 data   0,   0,   0,   0,   0,   0,   0,   0
II  1210 data   0,   0,   0,   0,   0,   0,   0,   0
CJ  1220 data   0,   0,   0,   0,   0,   0,   0,   0
MJ  1230 data   0,   0,   0,   0,   0,   0,   0,   0
GK  1240 data   0,   0,   0,   0,   0,   0,   0,   0
AL  1250 data   0,   0,   0,   0,   0,   0,   0,   0
KL  1260 data   0,   0,   0,   0,   0,   0,   0,   0
EM  1270 data   0,   0,   0,   0,   0,   0,   0,   0
OM  1280 data   0,   0,   0,   0,   0,   0,   0,   0
IN  1290 data   0,   0,   0,   0,   0,   0,   0,   0
CO  1300 data   0,   0,   0,   0,   0,   0,   0,   0
MO  1310 data   0,   0,   0,   0,   0,   0,   0,   0
GP  1320 data   0,   0,   0,   0,   0,   0,   0,   0
AA  1330 data   0,   0,   0,   0,   0,   0,   0,   0
KA  1340 data   0,   0,   0,   0,   0,   0,   0,   0
EB  1350 data   0,   0,   0,   0,   0,   0,   0,   0
MD  1360 data   0,   0,   0,   0,   0,   0,  32,  15
DM  1370 data 104, 169, 124, 133,  79, 169,  94, 133
PF  1380 data  78, 173,  90, 104, 208,  52,  32,  30
DM  1390 data 104,  32,  27, 104, 173,  90, 104, 208
JM  1400 data  41, 166,  78, 181,   0, 240,   8, 166
KM  1410 data  79, 181,   0, 240,  13, 208,  27,  32
GG  1420 data  60, 104, 240,  14, 166,  79, 181,   0
HF  1430 data 208, 220, 165,  78, 134,  78, 133,  79
AJ  1440 data 208, 212,  32,  42, 104,  32,  66, 104
FA  1450 data 208, 204, 166,  79, 181,   8, 133,  80
GM  1460 data 181,   9, 133,  81, 173,  90, 104, 201
FF  1470 data 253, 208,  40, 162, 124, 181,  16,  21
JN  1480 data  17, 240,  25, 174,  96, 104,  32, 201
PM  1490 data 255, 162,   0, 189, 170, 105, 240,   6
LB  1500 data 157, 194, 104, 232, 208, 245,  32,  57
FD  1510 data 104,  32, 204, 255, 169,   0, 141,  90
CP  1520 data 104, 240, 191,  32,  42, 104,  32,  18
IF  1530 data 104,  96,  13,  13, 109,  73,  83,  77
JJ  1540 data  65,  84,  67,  72,  69,  68,  32,  82
FL  1550 data  69,  67,  79,  82,  68,  83,  32,  79
CM  1560 data  86,  69,  82,  70,  76,  79,  87,  69
CJ  1570 data  68,  32,  66,  85,  70,  70,  69,  82
EJ  1580 data  46,  13,  98,  85,  70,  70,  69,  82
NM  1590 data  32,  67,  79,  78,  84,  69,  78,  84
FL  1600 data  83,  32,  87,  73,  76,  76,  32,  66
MM  1610 data  69,  32,  80,  82,  73,  78,  84,  69
PM  1620 data  68,  32,  65,  78,  68,  13,  84,  72
KO  1630 data  69,  32,  67,  79,  77,  80,  65,  82
CO  1640 data  73,  83,  79,  78,  32,  87,  73,  76
EA  1650 data  76,  32,  67,  79,  78,  84,  73,  78
HJ  1660 data  85,  69,  46,  13,   0,  32,  15, 104
EA  1670 data 173,  90, 104, 208, 123, 162,  79, 169
JH  1680 data 207,  32,  21, 104, 176, 109, 160,   3
DM  1690 data 177, 251, 133, 124, 200, 177, 251, 133
CL  1700 data 125, 162, 207, 169, 217,  32,  21, 104
OB  1710 data 176,  89, 165, 251, 133, 126, 165, 252
BM  1720 data 133, 127, 169,   0, 160,   3, 145, 251
HF  1730 data 169, 194,  32,  21, 104, 176,  68, 165
JP  1740 data 251, 133, 128, 165, 252, 133, 129, 162
NC  1750 data  78, 169, 206,  32,  21, 104, 176,  51
CM  1760 data 160,   3, 177, 251, 133, 130, 200, 177
EN  1770 data 251, 133, 131, 162, 206, 169, 217,  32
KP  1780 data  21, 104, 176,  31, 165, 251, 133, 132
AB  1790 data 165, 252, 133, 133, 169,   0, 160,   3
CE  1800 data 145, 251, 169, 194,  32,  21, 104, 176
KD  1810 data  10, 165, 251, 133, 134, 165, 252, 133
BI  1820 data 135, 144,   5, 169, 254, 141,  90, 104
OM  1830 data 173,  90, 104, 208,  75,  32, 204, 255
EJ  1840 data 162,  94, 134,  78, 169,  18, 149,  16
CH  1850 data 169,   1, 149,  17, 169,  65, 149,   1
OJ  1860 data 169,  70, 149,   2, 169,  30, 149,   3
KA  1870 data 169, 255, 149,  14, 166,  78, 180,  10
FE  1880 data 136, 148,   8, 180,  11, 136, 148,   9
ED  1890 data  32,  30, 104, 166,  78, 181,   0, 208
PF  1900 data  23, 181,   5,  41,   7, 240,   6,  32
FI  1910 data  36, 104,  32,  36, 104,  32, 236, 106
LM  1920 data  32,  27, 104, 173,  90, 104, 240, 212
AJ  1930 data  32,  18, 104,  96, 181,  10, 133, 251
CN  1940 data 181,  11, 133, 252, 169, 153, 133, 253
FD  1950 data 169, 104, 133, 254, 162,  33, 160,   0
FM  1960 data  32,  69, 104, 173, 156, 104,  16,  83
ME  1970 data  41, 127, 170, 160,   3, 177, 126, 208
GO  1980 data  28, 160,  15, 185, 159, 104, 209, 124
DP  1990 data 208,  19, 136,  16, 246, 160,   3, 138
PO  2000 data 145, 126, 173, 158, 104, 145, 128, 136
KA  2010 data 173, 157, 104, 145, 128, 160,   3, 177
GP  2020 data 132, 208,  31, 160,  15, 185, 159, 104
NP  2030 data 209, 130, 208,  31, 136,  16, 246, 160
LO  2040 data   3, 138, 145, 132, 173, 158, 104, 145
AP  2050 data 134, 136, 173, 157, 104, 145, 134,  24
NC  2060 data 144,   9, 177, 126, 240,   5, 169,   1
OD  2070 data 141,  90, 104,  96,  32,  15, 104, 173
KH  2080 data  90, 104, 240,   4,  32,  18, 104,  96
AO  2090 data  32, 204, 255, 162,  94, 134,  78, 169
CJ  2100 data  18, 149,  16, 169,   0, 149,  17, 169
AL  2110 data  65, 149,   1, 169,  70, 149,   2, 169
FE  2120 data  30, 149,   3, 169, 255, 149,  14,  32
LH  2130 data  36, 104, 173,  90, 104, 208, 124, 166
LM  2140 data  78, 169, 142, 149,  14,  32,  30, 104
BJ  2150 data 173,  90, 104, 208, 110,  32, 105, 108
MK  2160 data 160,  17, 185, 156, 104, 153, 204, 104
EL  2170 data 136,  16, 247, 173, 174, 104, 141, 224
```

```
IJ  2180 data 104, 173, 175, 104, 141, 225, 104, 169
NJ  2190 data  13, 141, 231, 104, 141, 232, 104, 169
AL  2200 data 147, 174,  96, 104, 224,   3, 240,   2
MM  2210 data 169,  12, 141, 194, 104,  32, 201, 255
NP  2220 data 162,  39,  32,  57, 104, 166,  78, 169
PM  2230 data   0, 149,   5, 169, 255, 149,  14, 166
LM  2240 data  78, 180,  10, 136, 148,   8, 180,  11
HN  2250 data 136, 148,   9,  32,  30, 104, 166,  78
AO  2260 data 181,   0, 208,  23, 181,   5,  41,   7
KO  2270 data 240,   6,  32,  36, 104,  32,  36, 104
LM  2280 data  32,  15, 108,  32,  27, 104, 173,  90
NN  2290 data 104,  16, 212,  32,  18, 104,  96,  32
DH  2300 data 105, 108, 173, 156, 104,  16,  81,  41
LA  2310 data 127,  10,  10, 105,   3, 168, 162,   3
DB  2320 data 185, 131, 108, 157, 216, 104, 136, 202
DH  2330 data  16, 246, 162,  15, 189, 159, 104, 157
LF  2340 data 199, 104, 202,  16, 247, 173, 185, 104
GF  2350 data 170, 173, 184, 104,  32,  84, 104, 160
EF  2360 data   5, 185,  87,   0, 153, 219, 104, 136
PM  2370 data  16, 247, 160,   3, 185, 131, 108, 153
EJ  2380 data 226, 104, 136,  16, 247, 169,  13, 141
LJ  2390 data 232, 104,  32, 204, 255, 174,  96, 104
IJ  2400 data  32, 201, 255, 162,  39,  32,  57, 104
IA  2410 data  96, 166,  78, 181,  10, 133, 251, 181
KM  2420 data  11, 133, 252, 169, 153, 133, 253, 169
EH  2430 data 104, 133, 254, 160,   0, 162,  33,  32
OH  2440 data  69, 104,  96,  66,  76,  75,  83,  83
NP  2450 data  69,  81,  32,  80,  82,  71,  32,  85
GH  2460 data  83,  82,  32,  82,  69,  76,  32, 169
GI  2470 data   0, 141,  91, 104, 141,  92, 104, 141
MB  2480 data  90, 104, 141,  26, 208, 141,  13, 221
CP  2490 data 169,  32, 162,  99, 157, 194, 104, 202
EJ  2500 data  16, 250, 162, 208, 169, 206,  32,  21
CP  2510 data 104, 176,   7, 160,   3, 177, 251, 141
AN  2520 data  96, 104, 162, 196, 169, 206,  32,  21
GA  2530 data 104, 176,   7, 160,   3, 177, 251, 141
FO  2540 data  93, 104, 169, 124, 133, 251, 169,   0
EC  2550 data 133, 252, 169, 135, 133, 253, 169, 104
KC  2560 data 133, 254, 162,  18, 160,   0,  32,  69
OE  2570 data 104, 173,  94, 104, 174,  93, 104, 172
IF  2580 data  94, 104,  32, 186, 255, 169,   2, 162
AN  2590 data  58, 160, 109,  32, 189, 255,  32, 192
DF  2600 data 255,  32,  39, 104, 208,  51, 173,  95
DN  2610 data 104, 174,  93, 104, 172,  95, 104,  32
KL  2620 data 186, 255, 169,   1, 162,  60, 160, 109
GJ  2630 data  32, 189, 255,  32, 192, 255,  32,  39
CE  2640 data 104, 208,  22, 160,  94, 173, 131, 104
PI  2650 data 162,  79,  32,  61, 109, 208,  10, 160
OP  2660 data 124, 173, 133, 104, 162,  78,  32,  61
LJ  2670 data 109,  96,  73,  48,  35, 141, 126, 104
GI  2680 data 169, 198,  32,  24, 104, 201,   0, 208
LF  2690 data   3, 173, 126, 104,  24, 153,  11,   0
NO  2700 data 109, 134, 104, 153,  13,   0, 237, 134
PL  2710 data 104, 153,   9,   0, 169, 214,  32,  24
PK  2720 data 104, 153,   1,   0, 169, 212,  32,  24
CM  2730 data 104, 153,   2,   0, 169, 218,  32,  24
IM  2740 data 104, 153,   3,   0, 169, 211,  32,  24
PM  2750 data 104, 153,   4,   0, 138,   9, 128, 170
LM  2760 data 169, 194,  32,  21, 104, 152, 170, 160
OC  2770 data   2, 177, 251, 149,  16, 200, 177, 251
LL  2780 data 149,  17, 169,   0, 149,   0, 149,   5
MD  2790 data 149,   6, 149,   7, 149,  15, 149,  10
II  2800 data 149,  12, 169, 255, 149,   8, 169,  66
LI  2810 data 213,   1, 240,   6, 169,  68, 213,   1
EC  2820 data 208,  12, 134,  78,  32,  36, 104, 149
OH  2830 data   5,  32,  36, 104, 149,   6, 173,  90
MI  2840 data 104,  96,  32, 204, 255, 173,  95, 104
BO  2850 data  32, 195, 255, 173,  94, 104,  32, 195
BE  2860 data 255, 162, 205, 169, 205,  32,  21, 104
HG  2870 data 176,  13, 160,   3, 173,  91, 104, 145
KG  2880 data 251, 136, 173,  92, 104, 145, 251, 162
LG  2890 data 210, 169, 195,  32,  21, 104, 176,   7
AL  2900 data 160,   3, 173,  90, 104, 145, 251, 169
GH  2910 data 124, 133, 253, 169,   0, 133, 254, 169
NI  2920 data 135, 133, 251, 169, 104, 133, 252, 162
FF  2930 data  18, 160,   0,  32,  69, 104, 169,   0
HE  2940 data 133, 198,  96, 134,  87, 133,  88, 152
CK  2950 data  72, 160,   0, 165,  45, 166,  46, 134
EF  2960 data 252, 133, 251, 228,  48, 208,   4, 197
CD  2970 data  47, 240,  25, 165,  87, 209, 251, 208
AK  2980 data   8, 165,  88, 200, 209, 251, 240,  15
GF  2990 data 136,  24, 165, 251, 105,   7, 144, 225
JP  3000 data 230, 252, 208, 221,  56, 176,   1,  24
OD  3010 data 104, 168, 166,  87, 165,  88,  96,  32
KO  3020 data  21, 104, 169,   0, 176,  24, 132,  87
EC  3030 data 160,   2, 177, 251, 240,  14, 200, 177
PA  3040 data 251, 133, 253, 200, 177, 251, 133, 254
BC  3050 data 160,   0, 177, 253, 164,  87,  96, 165
DL  3060 data 198, 240,  35, 169,   0, 133, 198, 173
MM  3070 data 119,   2,  72, 201, 134, 240,  17, 201
HL  3080 data 136, 208,  18, 104, 205, 119,   2, 208
LK  3090 data 234,  72, 169,   0, 133, 198, 240, 243
PM  3100 data 169, 255, 141,  90, 104, 104,  96,  32
GG  3110 data  33, 104, 166,  78, 173,  90, 104,  48
NJ  3120 data  68, 181,   7, 240,  64, 165, 253, 149
IF  3130 data   8, 165, 254, 149,   9, 160,   0, 181
HD  3140 data   5, 145, 253, 200, 181,   6, 145, 253
CL  3150 data 200, 181,   7, 145, 253,  56, 101, 253
NE  3160 data 133, 253, 169,   0, 101, 254, 133, 254
DN  3170 data 181,  12, 229, 253, 133, 253, 181,  13
JO  3180 data 229, 254, 133, 254, 144,  10, 201,   0
NK  3190 data 208,  11, 165, 253, 201,  85, 176,   5
MD  3200 data 169, 253, 141,  90, 104,  96, 164,  78
KN  3210 data 185,   8,   0, 133, 253, 185,   9,   0
KL  3220 data 133, 254, 162, 253,  32,  63, 104, 182
FK  3230 data   1, 224,  66, 208,  28,  32,  36, 104
FL  3240 data  32,  36, 104,  32,  36, 104, 153,   5
AM  3250 data   0,  32,  36, 104, 153,   6,   0, 185
PD  3260 data   0,   0, 240,  94, 162,   0, 240, 127
CD  3270 data  96, 185,   7,   0, 224,  65, 208,   4
DM  3280 data 169,   1, 208,  18, 224,  88, 208,  14
BG  3290 data 182,   2, 224,  67, 208,   6, 201,   0
MD  3300 data 240,   2, 105,   0, 182,   1,  24, 121
DC  3310 data   5,   0, 153,   5,   0, 144,   8, 169
IM  3320 data   0, 121,   6,   0, 153,   6,   0, 224
EF  3330 data  68, 208,  39, 162,   0,  32,  36, 104
LC  3340 data 176,  69, 160,   3, 145, 253,  32,  72
IM  3350 data 104, 162,   0, 161, 251,  41,   3, 133
OB  3360 data 251, 162,   1, 228, 251, 240,  48,  32
EE  3370 data  36, 104, 176,  43, 200, 145, 253, 232
PC  3380 data 208, 241, 182,   3, 142, 126, 104, 182
LD  3390 data   2, 224,  76, 208,  42, 162,   0, 160
AF  3400 data   3,  32,  36, 104, 176,  17, 145, 253
LD  3410 data 200, 232, 236, 126, 104, 208, 242, 205
OF  3420 data 126, 104, 141, 126, 104,  16,  24, 164
NP  3430 data  78, 150,   7,  96, 164,  78, 169, 254
GK  3440 data 141,  90, 104, 153,   0,   0,  96, 224
JO  3450 data  70, 208,  20, 160,   3, 162,   0,  32
AM  3460 data  36, 104, 176, 227, 145, 253, 200, 232
IL  3470 data 236, 126, 104, 144, 242, 176, 216, 185
PI  3480 data   4,   0, 141, 127, 104, 224,  67, 208
NB  3490 data  28, 160,   3, 162,   0,  32,  36, 104
```

```
JN 3500 data 176, 197, 145, 253, 200, 236, 127, 104
EJ 3510 data 144,   5, 205, 126, 104, 240, 184, 232
LM 3520 data 208, 235, 202, 208, 178, 224,  77, 208
JO 3530 data 179, 160,   3, 162,   0,  32,  36, 104
DP 3540 data 176, 165, 145, 253, 200, 236, 127, 104
LD 3550 data 144,   8,  45, 126, 104, 205, 126, 104
IM 3560 data 240,   6, 232, 208, 232, 202, 208, 143
PB 3570 data 232, 208, 140, 138,  72, 152,  72, 164
CK 3580 data  78,  56, 185,   0,   0, 208,  59, 185
LF 3590 data  14,   0, 217,  15,   0, 144,  10, 185
LD 3600 data  16,   0, 240,  40,  32,  89, 112, 176
HB 3610 data  41,  24, 185,  12,   0, 121,  14,   0
BF 3620 data 133,  87, 185,  13,   0, 105,   0, 133
AH 3630 data  88,  32,  75, 104, 162,   0, 161,  87
BL 3640 data 133,  87,  32,  75, 104, 152, 170, 246
NE 3650 data  14,  24, 144,   6, 169,   1, 153,   0
DE 3660 data   0,  56, 104, 168, 104, 170, 165,  87
AH 3670 data  96,  32,  81, 104,  32,  78, 104, 142
KM 3680 data 104, 104, 141, 105, 104, 185,  17,   0
KG 3690 data  32,  81, 104,  32,  78, 104, 142, 107
LG 3700 data 104, 141, 108, 104, 152,  72,  32, 204
EK 3710 data 255, 174,  94, 104,  32, 201, 255, 160
OJ 3720 data   0, 185,  97, 104, 240,   6,  32, 168
FN 3730 data 255, 200, 208, 245,  32, 174, 255,  32
JN 3740 data  39, 104, 208,  80, 174,  95, 104,  32
FO 3750 data 198, 255, 104, 168,  32, 165, 255, 153
AA 3760 data  16,   0,  32, 183, 255, 208,  65,  32
DB 3770 data 165, 255, 153,  17,   0,  32, 183, 255
JO 3780 data 208,  54, 169,   0, 153,  14,   0, 162
HB 3790 data 254, 217,  16,   0, 208,   3, 182,  17
CL 3800 data 202, 150,  15, 152,  72, 185,  12,   0
DB 3810 data 133,  89, 185,  13,   0, 133,  90, 134
HD 3820 data  87, 160,   0,  32, 165, 255, 145,  89
MB 3830 data  32, 183, 255,  41, 191, 208,   5, 200
EJ 3840 data 196,  87, 208, 239, 170, 104, 168, 138
LL 3850 data 153,   0,   0,  32, 171, 255, 185,   0
MI 3860 data   0, 240,   1,  56,  96,  32, 183, 255
CL 3870 data 208,  53, 174,  94, 104,  32, 198, 255
HH 3880 data 160,   0,  32, 165, 255, 153, 194, 104
BD 3890 data 200, 201,  13, 208, 245, 140, 193, 104
FK 3900 data  32, 204, 255, 173, 194, 104,  13, 195
HM 3910 data 104, 201,  48, 208,  12, 169,  32, 160
AG 3920 data 100, 136, 153, 194, 104, 208, 250, 240
GE 3930 data   6, 174, 193, 104,  32,  57, 104,  96
GI 3940 data 166,  78,  56, 181,  10, 245,   8, 133
KK 3950 data  82, 181,  11, 245,   9, 144,  21,  56
JM 3960 data 166,  79, 181,  10, 229,  80, 133,  83
PJ 3970 data 181,  11, 229,  81, 144,   6, 165,  82
CD 3980 data 197,  83, 240,  73, 238,  91, 104, 208
PG 3990 data   3, 238,  92, 104, 174,  96, 104,  32
AG 4000 data 201, 255, 166,  78, 224,  94, 208,  25
DF 4010 data  32, 216, 113, 162, 124, 181,  16,  21
EK 4020 data  17, 240,  39, 173,  90, 104,  48,  31
KB 4030 data  32, 181, 113,  32, 230, 113,  76, 151
NP 4040 data 113,  32, 230, 113, 162, 124, 181,  16
IL 4050 data  21,  17, 240,  14, 173,  90, 104,  48
BJ 4060 data   6,  32, 181, 113,  32, 216, 113,  32
NJ 4070 data 169, 113,  32, 204, 255,  96, 169,  13
BA 4080 data 141, 194, 104, 162,   1,  32,  57, 104
KD 4090 data  96,  32, 158, 113, 169,  61,  32, 183
PK 4100 data 113,  32, 158, 113,  96, 169,  45, 162
II 4110 data  79, 157, 194, 104, 202, 208, 250, 169
MM 4120 data  32, 141, 194, 104, 169,  13, 141,  18
ND 4130 data 105, 162,  81, 173,  96, 104, 201,   3
IF 4140 data 208,   2, 162,  40,  32,  57, 104,  96
JM 4150 data 166,  78, 181,   8, 133,  85, 181,   9
CH 4160 data 133,  86,  32, 244, 113,  96, 166,  79
PI 4170 data 165,  80, 133,  85, 165,  81, 133,  86
AI 4180 data  32, 244, 113,  96, 181,  10, 133,  82
IF 4190 data 181,  11, 133,  83, 181,   1, 141,  82
BG 4200 data 114,  32,  27, 104, 173,  90, 104,  48
DM 4210 data  58,  56, 165,  85, 229,  82, 165,  86
LA 4220 data 229,  83, 144,  47, 173,  82, 114, 201
FI 4230 data  65, 240,  26, 201,  66, 240,  17, 201
GO 4240 data  68, 240,   8,  32, 158, 113,  32,  54
MI 4250 data 104, 240,  16,  32,  51, 104, 208,  11
HJ 4260 data  32,  48, 104, 240,   6,  32, 158, 113
KI 4270 data  32,  45, 104, 162,  82,  32,  63, 104
JL 4280 data  76,   1, 114, 173,  82, 114, 201,  66
BN 4290 data 240,   7, 201,  68, 240,   3,  32, 158
JB 4300 data 113,  96,   0, 173,   8, 115,  72, 173
CM 4310 data 100, 114, 141,   8, 115,  32,  48, 104
LN 4320 data 104, 141,   8, 115,  96,  32, 208, 114
ON 4330 data  32,  75, 104, 160,   1, 177,  82, 170
AA 4340 data 136, 177,  82,  32,  75, 104,  32,  84
BA 4350 data 104, 160,   5, 185,  87,   0, 153, 194
AC 4360 data 104, 136,  16, 247, 160,   2,  32,  75
IH 4370 data 104, 177,  82,  32,  75, 104, 141, 200
FP 4380 data 114, 141, 202, 114, 200, 173, 200, 114
LF 4390 data 240,  45, 173, 205, 114, 133, 253, 173
CA 4400 data 203, 114, 141, 201, 114,  32, 247, 114
LI 4410 data  32,  87, 104,  32, 101, 115, 206, 202
JC 4420 data 114, 240,   5, 206, 201, 114, 208, 237
HH 4430 data 174, 204, 114, 134, 253, 169,  13,  32
KL 4440 data 101, 115,  32,  57, 104, 208, 206,  96
AD 4450 data   0,   0,   0,   0,   0,   0,   0,   0
PD 4460 data 169,   0, 141, 206, 114, 141, 207, 114
NK 4470 data 173,  96, 104, 201,   3, 240,   6, 169
LO 4480 data  64, 162,   8, 208,   4, 169,  32, 162
EL 4490 data   7, 141, 203, 114, 142, 205, 114,  24
LH 4500 data 109, 205, 114, 141, 204, 114,  96, 174
LL 4510 data 206, 114, 208,  38, 206, 200, 114,  32
PO 4520 data  75, 104, 177,  82,  32,  75, 104, 200
KK 4530 data 174, 207, 114, 208,  28, 201, 255, 240
DO 4540 data  24, 201, 128, 144,  20,  32,  56, 115
CL 4550 data 202, 138,  24, 109, 202, 114, 141, 202
EM 4560 data 114, 232, 202, 142, 206, 114, 189, 110
PD 4570 data 104, 201,  34, 208,  10,  72, 173, 207
DP 4580 data 114,  73,   1, 141, 207, 114, 104,  96
AE 4590 data  56, 233, 127, 170, 152,  72, 160, 255
DE 4600 data 169, 158, 133, 251, 169, 160, 133, 252
JC 4610 data 200, 177, 251,  16, 251, 202, 208, 248
LD 4620 data  41, 127, 157, 110, 104, 232, 136, 192
DI 4630 data 255, 240,   4, 177, 251,  16, 243, 142
GF 4640 data 206, 114, 104, 168,  96, 166, 253, 157
KN 4650 data 194, 104, 232, 134, 253,  96,  32,  75
BF 4660 data 104, 160,   5, 177,  82, 153,  87,   0
JD 4670 data 136,  16, 248,  32,  75, 104, 169,   1
MJ 4680 data 133, 253, 165,  88,  32, 187, 116, 165
FI 4690 data  87,  32, 187, 116, 169,  45,  32, 200
LN 4700 data 116, 169,   8, 133, 253, 164,  89, 165
EM 4710 data  90,  32, 187, 116, 136, 240,  23, 230
KO 4720 data 253, 165,  91, 141, 178, 116,  32, 187
PK 4730 data 116, 136, 240,  10, 230, 253, 165,  92
LP 4740 data 141, 179, 116,  32, 187, 116, 169,  20
HA 4750 data 133, 253, 165,  90,  32,  72, 104, 160
MN 4760 data   0, 177, 251, 133,  93, 200, 177, 251
MC 4770 data  32, 200, 116, 192,   3, 208, 246, 230
CB 4780 data 253, 165,  89, 201,   1, 240,  85, 169
AK 4790 data   8,  36,  93, 208,  18, 169,  48,  37
ON 4800 data  93, 240,  86, 201,  16, 240,  21, 165
HB 4810 data  89, 201,   2, 240,  81, 208,  49, 169
```

```
FI 4820 data  40,  32, 200, 116, 165,  89, 201,   2
AP 4830 data 240,  68, 208,  36,  24, 169,   2, 101
NC 4840 data  87, 133,  87, 169,   0, 101,  88, 133
EC 4850 data  88, 165,  87, 101,  91,  36,  91, 133
CM 4860 data  91,  16,   7, 165,  88, 233,   0, 133
BF 4870 data  88,  24, 165,  88, 105,   0, 133,  92
NB 4880 data 169,  36,  32, 200, 116, 165,  92,  32
ME 4890 data 187, 116, 208,  23, 169,   4,  36,  93
PL 4900 data 240,  80, 169,  65,  32, 200, 116, 208
IG 4910 data  73, 169,  35,  32, 200, 116, 169,  36
HM 4920 data  32, 200, 116, 165,  91,  32, 187, 116
JF 4930 data 169,  48,  37,  93, 201,  48, 208,  16
CG 4940 data 169,  64,  36,  93, 240,  23, 169,  44
GP 4950 data  32, 200, 116, 169,  88,  32, 200, 116
LO 4960 data 169,   8,  36,  93, 240,  28, 169,  41
GH 4970 data  32, 200, 116, 208,  21, 169,   8,  36
KH 4980 data  93, 240,   5, 169,  41,  32, 200, 116
PJ 4990 data 169,  44,  32, 200, 116, 169,  89,  32
AO 5000 data 200, 116, 169,  34, 133, 253, 169,  60
AN 5010 data  32, 200, 116, 164,  89, 165,  90,  32
PM 5020 data 180, 116, 136, 240,  15, 173, 178, 116
FF 5030 data  32, 180, 116, 136, 240,   6, 173, 179
AF 5040 data 116,  32, 180, 116, 169,  62,  32, 200
FL 5050 data 116, 169,  13,  32, 200, 116,  32,  57
MN 5060 data 104,  96,   0,   0,  32,  87, 104,  32
KP 5070 data 200, 116,  96,  32,  78, 104,  72, 138
KD 5080 data  32, 200, 116, 104,  32, 200, 116,  96
CC 5090 data 166, 253, 157, 194, 104, 232, 134, 253
NO 5100 data  96,  32,  98, 117, 160,   1, 132, 253
NC 5110 data  32, 171, 117, 160,   0,  32, 171, 117
ID 5120 data 160,   2,  32,  75, 104, 177,  82,  32
MH 5130 data  75, 104, 141,  93, 117, 200, 140,  92
NN 5140 data 117, 172,  92, 117, 169,   6, 133, 253
DJ 5150 data 173,  93, 117, 240,  94,  56, 237,  95
BL 5160 data 117, 170, 176,   8, 174,  93, 117, 142
EJ 5170 data  95, 117, 162,   0, 142,  93, 117, 174
GN 5180 data  95, 117, 142,  94, 117, 230, 253,  32
JJ 5190 data 137, 117, 173,  94, 117, 208, 246, 174
JD 5200 data  96, 117, 134, 253, 169,  60,  32, 209
JL 5210 data 117, 174,  95, 117, 142,  94, 117, 173
DA 5220 data  92, 117, 140,  92, 117, 168,  32, 193
IA 5230 data 117, 206,  94, 117, 208, 248, 174,  97
PJ 5240 data 117, 202, 134, 253, 169,  62,  32, 209
MJ 5250 data 117, 169,  13,  32, 209, 117,  32,  57
GO 5260 data 104, 169,  32, 157, 194, 104, 202,  16
HM 5270 data 250,  48, 150,  96,   0,   0,   0,   0
PC 5280 data   0,   0, 173,  96, 104, 201,   3, 240
NO 5290 data  16, 169,  16, 141,  95, 117, 169,  50
EJ 5300 data 141,  96, 117, 169,  68, 141,  97, 117
CC 5310 data  96, 169,   8, 141,  95, 117, 169,  28
CK 5320 data 141,  96, 117, 169,  38, 141,  97, 117
GB 5330 data  96, 173,  94, 117, 201,   5, 144,   3
PB 5340 data  32, 147, 117, 173,  94, 117, 201,   3
FN 5350 data 144,   3,  32, 157, 117, 230, 253,  32
MF 5360 data 171, 117, 206,  94, 117, 208,   1,  96
ID 5370 data 206,  94, 117,  32,  75, 104, 177,  82
LA 5380 data  32,  75, 104,  32,  78, 104,  72, 138
DD 5390 data  32, 209, 117, 104,  32, 209, 117, 200
PG 5400 data  96,  32,  75, 104, 177,  82,  32,  75
OH 5410 data 104,  32,  87, 104,  32, 209, 117, 200
AK 5420 data  96, 166, 253, 157, 194, 104, 232, 134
EI 5430 data 253,  96,  72, 138,  72, 152,  72, 160
BJ 5440 data   0, 185, 194, 104,  32, 210, 255, 169
JF 5450 data  32, 153, 194, 104, 200, 202, 208, 241
GL 5460 data 104, 168, 104, 170, 104,  96, 166,  78
KH 5470 data 181,   8, 133, 253, 181,   9, 133, 254
LI 5480 data 166,  79, 181,  10, 133,  80, 181,  11
OO 5490 data 133,  81, 181,   9, 197,  81,  48,  44
KM 5500 data 208,   6, 181,   8, 197,  80, 144,  36
MK 5510 data  32,  75, 104, 160,   2, 177, 253, 170
MD 5520 data 232, 177, 253, 209,  80, 208,   6, 202
KC 5530 data 240,  15, 200, 208, 244,  32,  75, 104
EM 5540 data 162,  80,  32,  63, 104, 166,  79, 208
PK 5550 data 209,  32,  75, 104,  96,  72, 152,  72
MI 5560 data 164,  78, 185,  10,   0,  24, 245,   0
IM 5570 data 208,   7, 181,   1, 217,  11,   0,  48
FB 5580 data  29, 181,   0, 133,  87, 181,   1, 133
FH 5590 data  88, 160,   2,  32,  75, 104, 177,  87
AI 5600 data  32,  75, 104,  24, 105,   3, 176,   8
JP 5610 data 117,   0, 208,   8, 240,   6, 169,   1
LC 5620 data  24, 117,   0,  56, 149,   0, 169,   0
GO 5630 data 117,   1, 149,   1, 104, 168, 104,  96
BG 5640 data 164,  79, 165,  80, 217,   8,   0, 208
LK 5650 data  19, 165,  81, 217,   9,   0, 208,  12
AC 5660 data 182,  10, 202, 150,   8, 182,  11, 202
OF 5670 data 150,   9, 208,  83, 162,  80,  32,  63
OG 5680 data 104, 165,  80, 133, 251, 165,  81, 133
OD 5690 data 252, 185,  10,   0, 133, 253, 185,  11
KI 5700 data   0, 133, 254, 185,   8,   0, 133,  80
BO 5710 data 185,   9,   0, 133,  81,  32,  63, 104
NJ 5720 data  56, 165,  80, 229, 251,  72, 165,  81
BJ 5730 data 229, 252,  72,  56, 165, 251, 229, 253
LB 5740 data 133,  87, 165, 252, 229, 254, 133,  88
IK 5750 data 185,   8,   0, 229,  87, 153,   8,   0
LL 5760 data 185,   9,   0, 229,  88, 153,   9,   0
KK 5770 data 104, 168, 104, 170,  32,  69, 104, 164
CI 5780 data  78, 182,  10, 202, 150,   8, 182,  11
KL 5790 data 202, 150,   9,  96, 134,  87, 132,  88
GM 5800 data 160,   0, 166,  88, 240,  20,  32,  75
AN 5810 data 104, 177, 251, 145, 253, 200, 208, 249
LI 5820 data  32,  75, 104, 230, 252, 230, 254, 202
DB 5830 data 208, 236, 165,  87, 240,  15,  32,  75
CP 5840 data 104, 177, 251, 145, 253, 200, 196,  87
NF 5850 data 208, 247,  32,  75, 104, 166,  87, 164
PE 5860 data  88,  96,  72, 169,  45, 133, 251, 169
IO 5870 data 120, 133, 252, 104,  72,  10, 144,   4
GH 5880 data 230, 252, 230, 252,  10, 144,   2, 230
NL 5890 data 252,  24, 101, 251, 133, 251, 169,   0
AF 5900 data 101, 252, 133, 252, 104,  96,   8,  72
LB 5910 data 165,   0,   9,   1, 133,   0, 165,   1
JB 5920 data  73,   1, 133,   1, 104,  40,  96,   8
OJ 5930 data 120, 248, 170,  24,  41,  15, 105, 144
BJ 5940 data 105,  64,  72, 138,  74,  74,  74,  74
ML 5950 data  24, 105, 144, 105,  64, 170, 104,  40
OH 5960 data  96,   8, 120, 248, 134,  87, 132,  88
DG 5970 data 170,  41,  15,  24, 105,   0, 168, 138
GI 5980 data  74,  74,  74,  74, 170, 232,  24, 152
AM 5990 data 202, 240,   4, 105,  22, 208, 249, 164
GO 6000 data  88, 166,  87,  40,  96, 133,  87, 134
GB 6010 data  88, 169,   0, 141, 127, 104, 141, 128
JF 6020 data 104, 141, 129, 104, 120, 248, 162,  16
DO 6030 data   6,  87,  38,  88, 173, 127, 104, 109
II 6040 data 127, 104, 141, 127, 104, 173, 128, 104
KJ 6050 data 109, 128, 104, 141, 128, 104, 173, 129
PH 6060 data 104, 109, 129, 104, 141, 129, 104, 202
NA 6070 data 208, 222, 216,  88, 169,  48, 141, 126
EJ 6080 data 104, 162,   0, 134,  93, 160,   3, 185
AN 6090 data 126, 104,  32,  78, 104,  72, 138,  32
GK 6100 data   4, 120, 104,  32,   4, 120, 136, 208
GE 6110 data 238, 169,  32, 197,  92, 208,   4, 169
OH 6120 data  48, 133,  92,  96, 205, 126, 104, 208
LH 6130 data   4, 169,  32, 208,   5, 162,   0, 142
```

```
KE 6140 data 126, 104, 166,  93, 149,  87, 232, 134
HA 6150 data  93,  96, 201,  32, 144,  12, 201, 128
HI 6160 data 144,  10, 201, 161, 144,   4, 201, 255
FF 6170 data 208,   2, 169,  46,  96,   1,  66,  82
HH 6180 data  75, 122,  79,  82,  65,   1,  63,  63
OI 6190 data  63,   1,  63,  63,  63,   1,  63,  63
MK 6200 data  63,  34,  79,  82,  65,  34,  65,  83
KK 6210 data  76,   1,  63,  63,  63,   1,  80,  72
KM 6220 data  80,   2,  79,  82,  65,   5,  65,  83
FM 6230 data  76,   1,  63,  63,  63,   1,  63,  63
KN 6240 data  63,  35,  79,  82,  65,  35,  65,  83
EK 6250 data  76,   1,  63,  63,  63,  18,  66,  80
FN 6260 data  76,  58,  79,  82,  65,   1,  63,  63
ON 6270 data  63,   1,  63,  63,  63,   1,  63,  63
FP 6280 data  63, 114,  79,  82,  65, 114,  65,  83
MB 6290 data  76,   1,  63,  63,  63,   1,  67,  76
JO 6300 data  67,  51,  79,  82,  65,   1,  63,  63
GA 6310 data  63,   1,  63,  63,  63,   1,  63,  63
CC 6320 data  63, 115,  79,  82,  65, 115,  65,  83
OP 6330 data  76,   1,  63,  63,  63,  35,  74,  83
OA 6340 data  82, 122,  65,  78,  68,   1,  63,  63
MP 6350 data  63,   1,  63,  63,  63,  34,  66,  73
JF 6360 data  84,  34,  65,  78,  68,  34,  82,  79
GF 6370 data  76,   1,  63,  63,  63,   1,  80,  76
KI 6380 data  80,   2,  65,  78,  68,   5,  82,  79
FD 6390 data  76,   1,  63,  63,  63,  35,  66,  73
HI 6400 data  84,  35,  65,  78,  68,  35,  82,  79
NF 6410 data  76,   1,  63,  63,  63,  18,  66,  77
OG 6420 data  73,  58,  65,  78,  68,   1,  63,  63
OH 6430 data  63,   1,  63,  63,  63,   1,  63,  63
PJ 6440 data  63, 114,  65,  78,  68, 114,  82,  79
JL 6450 data  76,   1,  63,  63,  63,   1,  83,  69
OI 6460 data  67,  51,  65,  78,  68,   1,  63,  63
GK 6470 data  63,   1,  63,  63,  63,   1,  63,  63
MM 6480 data  63, 115,  65,  78,  68, 115,  82,  79
CN 6490 data  76,   1,  63,  63,  63,   1,  82,  84
PL 6500 data  73, 122,  69,  79,  82,   1,  63,  63
OM 6510 data  63,   1,  63,  63,  63,   1,  63,  63
LO 6520 data  63,  34,  69,  79,  82,  34,  76,  83
NN 6530 data  82,   1,  63,  63,  63,   1,  80,  72
PC 6540 data  65,   2,  69,  79,  82,   5,  76,  83
KN 6550 data  82,   1,  63,  63,  63,  35,  74,  77
DB 6560 data  80,  35,  69,  79,  82,  35,  76,  83
PO 6570 data  82,   1,  63,  63,  63,  18,  66,  86
CB 6580 data  67,  58,  69,  79,  82,   1,  63,  63
OB 6590 data  63,   1,  63,  63,  63,   1,  63,  63
ME 6600 data  63, 114,  69,  79,  82, 114,  76,  83
PE 6610 data  82,   1,  63,  63,  63,   1,  67,  76
IB 6620 data  73,  51,  69,  79,  82,   1,  63,  63
GE 6630 data  63,   1,  63,  63,  63,   1,  63,  63
JH 6640 data  63, 115,  69,  79,  82, 115,  76,  83
FG 6650 data  82,   1,  63,  63,  63,   1,  82,  84
AF 6660 data  83, 122,  65,  68,  67,   1,  63,  63
OG 6670 data  63,   1,  63,  63,  63,   1,  63,  63
HI 6680 data  63,  34,  65,  68,  67,  34,  82,  79
JI 6690 data  82,   1,  63,  63,  63,   1,  80,  76
CN 6700 data  65,   2,  65,  68,  67,   5,  82,  79
HH 6710 data  82,   1,  63,  63,  63,  43,  74,  77
PK 6720 data  80,  35,  65,  68,  67,  35,  82,  79
PI 6730 data  82,   1,  63,  63,  63,  18,  66,  86
JK 6740 data  83,  58,  65,  68,  67,   1,  63,  63
OL 6750 data  63,   1,  63,  63,  63,   1,  63,  63
NN 6760 data  63, 114,  65,  68,  67, 114,  82,  79
MO 6770 data  82,   1,  63,  63,  63,   1,  83,  69
JL 6780 data  73,  51,  65,  68,  67,   1,  63,  63
GO 6790 data  63,   1,  63,  63,  63,   1,  63,  63

KA 6800 data  63, 115,  65,  68,  67, 115,  82,  79
MP 6810 data  82,   1,  63,  63,  63,   1,  63,  63
AO 6820 data  63, 122,  83,  84,  65,   1,  63,  63
IO 6830 data  63,   1,  63,  63,  63,  34,  83,  84
AE 6840 data  89,  34,  83,  84,  65,  34,  83,  84
CG 6850 data  88,   1,  63,  63,  63,   1,  68,  69
HG 6860 data  89,   1,  63,  63,  63,   1,  84,  88
KB 6870 data  65,   1,  63,  63,  63,  35,  83,  84
OG 6880 data  89,  35,  83,  84,  65,  35,  83,  84
FE 6890 data  88,   1,  63,  63,  63,  18,  66,  67
NE 6900 data  67,  58,  83,  84,  65,   1,  63,  63
LE 6910 data  63,   1,  63,  63,  63, 114,  83,  84
NN 6920 data  89, 114,  83,  84,  65,  50,  83,  84
MK 6930 data  88,   1,  63,  63,  63,   1,  84,  89
LH 6940 data  65,  51,  83,  84,  65,   1,  84,  88
MI 6950 data  83,   1,  63,  63,  63,   1,  63,  63
FH 6960 data  63, 115,  83,  84,  65,   1,  63,  63
JL 6970 data  63,   1,  63,  63,  63,   2,  76,  68
KL 6980 data  89, 122,  76,  68,  65,   2,  76,  68
LK 6990 data  88,   1,  63,  63,  63,  34,  76,  68
CO 7000 data  89,  34,  76,  68,  65,  34,  76,  68
MO 7010 data  88,   1,  63,  63,  63,   1,  84,  65
NB 7020 data  89,   2,  76,  68,  65,   1,  84,  65
FN 7030 data  88,   1,  63,  63,  63,  35,  76,  68
AB 7040 data  89,  35,  76,  68,  65,  35,  76,  68
FO 7050 data  88,   1,  63,  63,  63,  18,  66,  67
EO 7060 data  83,  58,  76,  68,  65,   1,  63,  63
MO 7070 data  63,   1,  63,  63,  63, 114,  76,  68
HJ 7080 data  89, 114,  76,  68,  65,  50,  76,  68
HE 7090 data  88,   1,  63,  63,  63,   1,  67,  76
CB 7100 data  86,  51,  76,  68,  65,   1,  84,  83
BD 7110 data  88,   1,  63,  63,  63, 115,  76,  68
DH 7120 data  89, 115,  76,  68,  65, 115,  76,  68
AG 7130 data  88,   1,  63,  63,  63,   2,  67,  80
OE 7140 data  89, 122,  67,  77,  80,   1,  63,  63
EB 7150 data  63,   1,  63,  63,  63,  34,  67,  80
MH 7160 data  89,  34,  67,  77,  80,  34,  68,  69
GI 7170 data  67,   1,  63,  63,  63,   1,  73,  78
EM 7180 data  89,   2,  67,  77,  80,   1,  68,  69
IF 7190 data  88,   1,  63,  63,  63,  35,  67,  80
KK 7200 data  89,  35,  67,  77,  80,  35,  68,  69
CI 7210 data  67,   1,  63,  63,  63,  18,  66,  78
MI 7220 data  69,  58,  67,  77,  80,   1,  63,  63
OJ 7230 data  63,   1,  63,  63,  63,   1,  63,  63
CC 7240 data  63, 114,  67,  77,  80,  50,  68,  69
NN 7250 data  67,   1,  63,  63,  63,   1,  67,  76
LJ 7260 data  68,  51,  67,  77,  80,   1,  63,  63
GM 7270 data  63,   1,  63,  63,  63,   1,  63,  63
AA 7280 data  63, 115,  67,  77,  80, 115,  68,  69
GP 7290 data  67,   1,  63,  63,  63,   2,  67,  80
ON 7300 data  88, 122,  83,  66,  67,   1,  63,  63
EL 7310 data  63,   1,  63,  63,  63,  34,  67,  80
NB 7320 data  88,  34,  83,  66,  67,  34,  73,  78
GC 7330 data  67,   1,  63,  63,  63,   1,  73,  78
DH 7340 data  88,   2,  83,  66,  67,   1,  78,  79
IN 7350 data  80,   1,  63,  63,  63,  35,  67,  80
LE 7360 data  88,  35,  83,  66,  67,  35,  73,  78
DC 7370 data  67,   1,  63,  63,  63,  18,  66,  69
NB 7380 data  81,  58,  83,  66,  67,   1,  63,  63
OD 7390 data  63,   1,  63,  63,  63,   1,  63,  63
IK 7400 data  63, 114,  83,  66,  67,  50,  73,  78
KH 7410 data  67,   1,  63,  63,  63,   1,  83,  69
GE 7420 data  68,  51,  83,  66,  67,   1,  63,  63
GG 7430 data  63,   1,  63,  63,  63,   1,  63,  63
CI 7440 data  63, 115,  83,  66,  67, 115,  73,  78
KH 7450 data  67,   1,  63,  63,  63,   0,   0
```

# PopToFront for the Amiga

**Bryce Nesbitt**
**Berkeley, CA**
**(C) 1987 Bryce Nesbitt**

Have you ever found a corner of a window you wished to work with, but had to play some sort of electronic shell game to bring it to the front? This utility provides a quick shortcut: simply activate any window in any screen by clicking into any part of it, hit left Amiga–F and that window will come forward. PopToFront is written entirely in 68000 for the Amiga. In addition to being a useful utility, the code provides a valuable example of adding a keyboard handler to the system and includes a version of the "exec_support" library, a set of routines normally available only to C programmers.

## Getting it running

The source code in listing 1 can be typed into any editor and assembled with the Metacomco 68000 assembler. (If you are lazy, a copy is available on the new Transactor Amiga disk). An icon can be created by modifying an existing icon of type "tool" using the editor in the system drawer of the workbench disk.

To activate PopToFront, double–click on its icon from Workbench or type 'run PopToFront' from the CLI. To test it out, push a window to the back with the back gadget, press and hold the Left Amiga key and tap "F" – the window will pop up to the front. Once installed, PopToFront stays resident until the Amiga is turned off. While it is possible to install several PopToFronts at once, no benefit results.

## How it works

The program consists of three parts: The exec_support routines, the startup code and the keyboard handler itself. The startup code and exec_support routines exist only to get the handler installed. This is done in three parts:

1) The intuition.library is opened and its address placed in the handler code for later use. Note that since the handler uses it, the intuition.library is never closed.
2) The startup code allocates some PUBLIC memory to contain the handler. PUBLIC is used so that, on a future version of the Amiga with a memory–management unit, there is no possibility that the handler will ever be swapped out to disk.
3) Once the handler is in its new home, the startup code adds it to the handler list by sending to the input.device an I/O Request block of type IND_ADDHANDLER. Its purpose thus fulfilled and errors checked for, it terminates, leaving the handler in place.

The handler gazes lightly into the meaning of each input.device event that passes by. When the correct one is detected it finds the pointer to the active Window & Screen from their known locations in the IntuitionBase structure, and calls the Intuition ScreenToFront( ) and WindowToFront( ) routines. The Left Amiga–F keystroke is then removed from the input stream.

```
;PopToFront (C)1987 Bryce Nesbitt. Unlimited free non-exclusive licence
;hereby granted to any sentient being to use or abuse this code in any way
;whatsoever provided that this and any other copyright notices remain fully
;attached and are reproduced in any simultaneously distributed printed matter
;and with the exception that, without prior written permission, it not
;be utilized by any entity that has been commonly referred to as Robert
;W. Skyles, Skyles Electric Works, Jim Drew, Regie Warren or any organization
;founded by, controlled, employing or profiting any such entity, its
;offspring or spouses.
;Author correspondence, bug or stupidity reports may be directed to:
;               Bryce Nesbitt
;               1712 Marin Ave.
;               Berkeley, Ca 94707-2902
*****************************
               NOLIST
               INCLUDE 'exec/types.i'
               INCLUDE 'exec/memory.i'
               INCLUDE 'exec/interrupts.i'
               INCLUDE 'exec/io.i'
               INCLUDE 'libraries/dosextens.i'
               INCLUDE 'devices/inputevent.s.i'
               INCLUDE 'devices/input.i'
               INCLUDE 'intuition/intuitionbase.i'
               INCLUDE 'workbench/startup.i'
               ;INCLUDE 'lib/exec_lib.i'     ;eliminates link with amiga.lib
               ;INCLUDE 'lib/dos_lib.i'      ;non-standard, and very speedy!
               ;INCLUDE 'lib/intuition_lib.i' ;See exec/exec_lib.i & fd.files
               LIST
jsrlib         macro
               xref    _LVO\1
               jsr     _LVO\1(a6)
               endm
jmplib         macro
               xref    _LVO\1
               jmp     _LVO\1(a6)
               endm
*****************************
               CODE
startup:       move.l  4,a6
               suba.l  a1,a1           ;Get THIS task
               jsrlib  FindTask
               move.l  d0,a5
               moveq   #0,d0           ;Set zero for later
               move.l  pr_CLI(a5),d1   ;Pointer to CLI only structure
               bne.s   fromCLI         ;If not zero, then save a zero. . .
;
; If called from Workbench a message will be sent. This waits for it,
; and saves its pointer to be returned to Workbench later.
;
               lea     pr_MsgPort(a5),a0
```

```
                jsrlib    WaitPort
                lea       pr_MsgPort(a5),a0
                jsrlib    GetMsg          ;Message pointer in D0
fromCLI         move.l    d0,-(a7)        ;Save message for later. . .
********************************* [A6=ExecBase][a5=this task]
failcode        equr      d7
portsave        equr      a5
IOReqsave       equr      a4

                moveq     #25,failcode    ;Default fail code
;---Prepare StdIO---
                moveq     #0,d0           ;priority
                bsr       _CreatePortE    ;[d0=Port]
                beq       ExitToDOS
                move.l    d0,portsave
                bsr       _CreateStdIOE   ;[a1=IoRequest]
                beq       e_StdIO
                move.l    a1,IOReqsave
;---Open input.device---
                moveq     #0,d0           ;Unit
                move.l    d0,d1           ;Flags
                lea       devname(pc),a0
                ;(IoRequest is in A1)
                jsrlib    OpenDevice      ;[d0=zero if ok]
                tst.l     d0
                bne       e_Open
;---Copy Handler---
                moveq     #0,d0           ;Any version
                lea       IntuiName(pc),a1
                jsrlib    OpenLibrary     ;[d0=Base]
                move.l    d0,IBASE+2      ;modify code
                beq.s     e_Intui

                move.l    #DownEnd-Start,d0
                move.l    #MEMF_PUBLIC,d1
                jsrlib    AllocMem
                move.l    d0,a2           ;location of block
                move.l    d0,a1
                tst.l     d0
                beq.s     e_nomem

                lea       Start(pc),a0
                moveq     #((DownEnd-Start)/4),d1  ;Length in LONGS
copylp          move.l    (a0)+,(a1)+     ;Copy handler to Public Memory
                dbra      d1,copylp
                ;68010 Loop mode!
;---Prepare IOReq with proper addresses---
                lea       Handlercode-Start(a2),a0  ;offset+memblock
                move.l    a0,IS_DATA(a2)  ;unused
                move.l    a0,IS_CODE(a2) ;where code is
                lea       PopName-Start(a2),a0     ;optional ascii name
                move.l    a0,LN_NAME(a2)
;---Send ADDHANDLER---
                move.l    IOReqsave,a1
                move.w    #IND_ADDHANDLER,IO_COMMAND(a1)
                move.l    a2,IO_DATA(a1)
                ;(IOReq in a1)
                jsrlib    DoIO            ;d0=(IOReq-a0)
                tst.l     d0
                bne.s     e_DoIO          ;zero=ok!
                moveq     #0,d7           ;set return code to zero
e_nomem
e_DoIO
e_Intui         move.l    IOReqsave,a1
                jsrlib    CloseDevice
e_Open          move.l    IOReqsave,a1
                bsr       _DeleteStdIOE
e_StdIO         move.l    portsave,a1
                bsr       _DeletePortE
********************************* (a7)+=message d7=return code
ExitToDOS       move.l    (a7)+,d6
                beq.s     NotWB           ;If saved pointer is zero, exit to CLI. . .
;
; Return the startup message to the parent Workbench tool. The forbid
; is needed so Workbench can't UnLoadSeg( ) the code too early.
;
                move.l    4,a6
                jsrlib    Forbid
                move.l    d6,a1           ;message pointer
                jsrlib    ReplyMsg
NotWB           move.l    d7,d0           ;Set "failat" code
                rts
IntuiName       dc.b      'intuition.library',0
devname         dc.b      'input.device',0
                cnop      0,2             ;WORD align

;**** exec_support/CreatePort ***
;
; port=_CreatePort(pri)
;  d0                d0.b
;
;FUNCTION: Create a nameless message port with a specified priority.
; (exec/ports.i)
;RESULT: The port pointer or Z=1 if an error occurred.
;REGISTERS: a6 is destroyed unless _CreatePortE is called,
; in which case a6 must contain ExecBase.
;EXAMPLE:
;               moveq     #3,d0           ;Set Priority
;               jsr       _CreatePort
;               beq.s     noport          ;Not enough memory or signals
;
                ;xref     _CreatePort
                ;xref     _CreatePortE
;_CreatePort    move.l    4,a6
_CreatePortE    move.l    a2,-(a7)
                move.b    d0,-(a7)
                move.l    #MEMF_PUBLIC+MEMF_CLEAR,d1
                moveq     #MP_SIZE,d0
                jsrlib    AllocMem
                move.l    d0,a2
                tst.l     d0
                beq.s     cp_nomemory
                moveq     #-1,d0
                jsrlib    AllocSignal     ;d0=return
                moveq     #-1,d1
                cmp.l     d0,d1           ;-1 indicates bad signal
                bne.s     cp_sigok
                move.l    a2,a1
                moveq     #MP_SIZE,d0
                jsrlib    FreeMem
cp_nomemory     addq.l    #2,a7           ;Unpush byte
                moveq     #0,d0           ;set z=1
                bra.s     cp_xit

cp_sigok        move.b    d0,MP_SIGBIT(a2)
                move.b    #PA_SIGNAL,MP_FLAGS(a2)
                move.b    #NT_MSGPORT,LN_TYPE(a2)
                move.b    (a7)+,LN_PRI(a2)
                suba.l    a1,a1           ;a1=0/Find this task
                jsrlib    FindTask        ;[d0=this task]
                move.l    d0,MP_SIGTASK(a2)
                lea       MP_MSGLIST(a2),a0  ;Point to list header
                NEWLIST   a0              ;Init new list macro
                move.l    a2,d0
cp_xit          move.l    (a7)+,a2        ;cc's NOT affected
                rts
;**** exec_support/DeletePort ***
;
;_DeletePort(port)
;               a1
;
;FUNCTION: Deletes the port by first setting some
; fields to illegal values then calling FreeMem.
;RESULT: none
;REGISTERS: a6 is destroyed unless _DeleteStdIOE is called,
; in which case a6 must contain ExecBase.
;
                ;xref     _DeletePort
                ;xref     _DeletePortE
;_DeletePort    move.l    4,a6
```

```
_DeletePortE  move.l   a1,-(a7)
              moveq    #-1,d0
              move.b   d0,LN_TYPE(a1)
              move.l   d0,MP_MSGLIST+LH_HEAD(a1)
              moveq    #0,d0              ;Clear upper 3/4 of d0
              move.b   MP_SIGBIT(a1),d0
              jsrlib   FreeSignal
              move.l   (a7)+,a1
              moveq    #MP_SIZE,d0
              jmplib   FreeMem
;
;ioStdReq=CreateStdIO(ioReplyPort)
;    a1                    d0
;
;Function: Allocate a IoRequest block of standard size.
;Example:      move.l   MyPort,d0
;             jsr      _CreateStdIO
;             beq.s    badnews
;
              ;xref    _CreateStdIOE
              ;xref    _CreateStdIO
;_CreateStdIO  move.l   4,a6
_CreateStdIOE move.l   d0,-(a7)
              moveq    #IOSTD_SIZE,d0
              move.l   #MEMF_PUBLIC+MEMF_CLEAR,d1
              jsrlib   AllocMem
              move.l   d0,a1
              tst.l    d0
              beq.s    nomem
              move.b   #NT_MESSAGE,LN_TYPE(a1)
              move.l   (a7)+,MN_REPLYPORT(a1)
nomem         rts

;••• exec_support/DeleteStdIO •••
;
;_DeleteStdIO(ioStdReq)
;              a1
;
;FUNCTION: Deletes an ioStdReq by setting some
; fields to illegal values then calling FreeMem.
;RESULT: none
;REGISTERS: a6 is destroyed unless _DeleteStdIOE is called,
; in which case a6 must contain ExecBase.
;
              ;xref    _DeleteStdIO
              ;xref    _DeleteStdIOE
;_DeleteStdIO  move.l   4,a6               ;Get ExecBase
_DeleteStdIOE moveq    #-1,d0             ;Set fields to illegal value
              move.b   d0,LN_TYPE(a1)
              move.l   d0,IO_DEVICE(a1)
              move.l   d0,IO_UNIT(a1)
              moveq    #IOSTD_SIZE,d0
              jmplib   FreeMem

;-------------------------------------------------------
              cnop     0,4               ;LONG word align
Start
ints          ;This is an interrupt structure (exec/interrupts.i)
              dc.l     0                 ;succ
              dc.l     0                 ;pred
              dc.b     0                 ;type
              dc.b     51                ;Priority (One step higher than Intuition)
              dc.l     0                 ;Name
              dc.l     0                 ;Data
              dc.l     0                 ;Code
inte

;When the handler is entered a0 will point to a linked list of input
; events of the type defined in the incude file (devices/inputevent.i)
; When done mangling the input stream it returns a new pointer in d0
;Note: All references to (a0) below refer to ie_NextEvent(a0). The
; offset equates to zero and was eliminated for efficiency.

Handlercode   move.l   a0,d0             ;Save pointer to start of chain
              cmpi.w   #$0100,ie_Class(a0)
;Compare class AND subclass. $0100 is a combination of IECLASS_RAWKEY
;for ie_Class and IECLASS_NULL for ie_SubClass.

              beq.s    hurdle1
              move.l   (a0),d1          ;Get }ie_NextEvent{ pointer
              bne.s    MoreEvents       ;If any. . .
              rts                       ;exit (with no changes made)

hurdle1       move.l   ie_Code(a0),d1   ;Code & Qualifier.
              andi.w   #%0000001111111011,d1

;Mask off the bits of the qualifier field that we do not care about. These
; are IEQUALIFER_RELATIVEMOUSE, _MULTIBROADCAST, _INTERRUPT,
; _CAPSLOCK, _LBUTTON, _RBUTTON and _MBUTTON

              cmpi.l   #$00230040,d1

;The upper word of this comparison ($0023) is the RAW key code for 'F'
; the lower is IEQUALIFIER_LCOMMAND.

              beq.s    hurdle2
KeepLooking   move.l   (a0),d1           ;See above. . .
              bne.s    MoreEvents       ;See above. . .
              rts                       ;exit (no changes)
;
;MoreEvents handles the case where several events may be linked and
; can extract one of them from the middle.
;
MoreEvents    move.l   a0,a1            ;Temporary for unlinking
              move.l   d1,a0            ;Look here next. . .
              cmpi.w   #$0100,ie_Class(a0)
              bne.s    KeepLooking
              move.l   ie_Code(a0),d1
              andi.w   #%0000001111111011,d1
              cmpi.l   #$00230040,d1
              bne.s    KeepLooking
;
;Move this event's ie_NextEvent pointer to that of the previous event,
; thereby unlinking this one from the chain.
;
              move.l   (a0),(a1)        ;Move pointer to NEXT to previous
              move.l   d0,-(a7)         ;Pointer to be passed back
              bra.s    hurdle3          ;Go do it. . .
;
;The pointer to the currently active Screen and Windows are available at a
; positive offset from the Intuition base pointer. These will be brought
; to the front. Note that it may not be proper under the Amiga system to
; do this in the actual handler. It may be safer to have another task
; lying around in the background to do the actual work. The description
; of the Intuition WindowToFront commands indicates that it will not take
; effect until the NEXT input event, which (of course) will not happen
; until this one exits.
;
hurdle2       move.l   (a0),-(a7)       ;Unlink first event. }ie_NextEvent{
hurdle3       move.l   a6,-(a7)
IBASE         move.l   #0,a6   ;Self-modifying-> Intuitionbase goes here
              move.l   ib_ActiveScreen(a6),d0
              beq.s    inoscreen
              move.l   d0,a0
              jsrlib   ScreenToFront
inoscreen     move.l   ib_ActiveWindow(a6),d0
              beq.s    inowindow
              move.l   d0,a0
              jsrlib   WindowToFront
inowindow     move.l   (a7)+,a6
              move.l   (a7)+,d0         ;Pointer to new, shorter input stream
              rts

PopName       dc.b     'PopToFront',0
              cnop     0,4              ;Pad out to Longword boundary
DownEnd
              END
```

# TrapSnapper: Adding A Trap Handler to Your C Programs

by Chris Zamara and Nick Sullivan

*That bug in your program could cost you a visit from the Guru. . .*
*TrapSnapper helps keep the old geezer at bay!*

If you've spent any amount of time with the Amiga, you are probably not unfamiliar with the following message: "Software Error – Task Held. Click on Cancel to Reset/Debug". This is an annoying message. If you accept the invitation to "Click on Cancel", you'll reset the machine, bringing down not only the task that failed but any others you may have going at the time. You can ignore the message, if you want (providing you have Workbench up or access to an extra CLI), but the "Software Error" requester keeps coming relentlessly back every time you swap disks or shuffle windows. Furthermore, all visible traces of the program that died – like its windows and screens – will still be around, getting in your way and using up memory.

There's a way to deal with software errors, at least in your own programs, but before we get into that let's look at what these errors are and why the Amiga handles them as it does.

### Tasks, Traps and Exceptions

As you probably know, the programs that you run on the Amiga are treated as separate tasks under Exec, which is the name given to the set of operating system routines that are responsible for multitasking, device, library and I/O management. Only one task can have control of the CPU at any one instant of course; that task is said to be in the "running" state. Other tasks will at the same time be either "waiting" for their turn at the CPU, or lying dormant ("sleeping") until they are awoken by an external event such as a mouse movement, keypress, or timer signal. Sleeping tasks require only a tiny proportion of processor time to service. If its wake–up event never comes, such a task could sleep for ever without perceptibly affecting the operation of the machine.

One difficulty with a multitasking system is that any task in the running state can crash due to a bug in the program. Since there may be other, viable, tasks in the system when this occurs, it is important that they should be allowed to continue unhindered if at all possible. This is where a Software Error differs from a Guru Meditation Error – Software Errors are polite bugs that don't step on outside tasks; Gurus are barbarian bugs that destroy everyone else along with themselves.

Software Errors are actually detected by the microprocessor itself, not by the operating system. In 68000 jargon, they are called "exceptions", for they are conditions that the processor cannot handle by ordinary means. An example is the undefined result of a division by zero, using one of the 68000 divide instructions DIVS or DIVU. If this operation is requested (usually by accident), the 68000 does not know how to proceed, and invokes an exception.

At this stage, several things happen. The 68000 goes into "supervisor" mode, which enables the status register (SR) and several privileged opcodes, and switches over to the supervisor stack from the user stack. Six bytes of data are pushed onto the supervisor stack: a word containing a copy of the Condition Code Register (CCR), and a long word containing the value of the program counter, which for most exceptions is the address of the instruction following the one that caused the exception. (In the case of certain exceptions, other data are also pushed, but we'll get to that later.) Finally, control is turned over to an exception–handling routine that is accessed via a table of vectors keyed to the "exception number" of the exceptional condition. For example, division by zero is exception 5; hence, its exception routine is entered through vector number 5 of the table. It is at this point that Exec takes over.

Exec pushes a long–word on the stack corresponding to the exception vector number – this is called the trap number. (In Amiga jargon, the term "exception" has been appropriated to another purpose, and exceptions are known as "traps", from the exception–generating TRAP and TRAPV instructions of the 68000 – see the list of trap numbers at the end of this article). It then branches through a vector that is specific to the task that caused the exception. Unless this vector has been changed by the task itself, it points to the default trap handler. Since Exec cannot on its own know what actions would be appropriate to take to rescue a given task from a given exception, its default handler does nothing more than put the task to sleep (permanently), and put up the "Software Error" requester to tell the user what is going on and provide him or her with the opportunity to reset if desired. Any resources (primarily RAM) allocated to the errant task remain allocated, and there is no reasonable way of getting them back.

However, it is not very difficult to create and enable your own trap–handling routine in a program that you write, and there are considerable advantages to doing so. For one thing, you can provide a graceful exit in the event that your program bombs out; more importantly, you can hand back your allocated RAM to the system, to be made available to other tasks. You can close any windows or screens that the program may have opened so that they don't continue to get in the way and use up memory. You also avoid the "Software Error" message that will otherwise be haunting you until you reset your machine. There is a slight risk involved: if the event that caused the trap had trashed your handler code before the 68000 intervened, you're off to Guru City; this risk is small enough to be acceptable.

We now have to consider how to write a handler routine that will get you through the trap and into your de-allocation code, and how to link this routine into the system so that it will be invoked when a trap occurs. We will outline what we believe to be the simplest approach to this problem in the following sections.

### Writing a Trap–Handler

As described above, the 68000's behaviour during a trap involves entering supervisor mode, pushing the CCR and the program counter, and jumping into the system handler code through a vector. If this were

the Commodore 64 rather than the Amiga, the obvious approach would be to change that vector to point to our own code. In a multitasking system, however, we can't take that kind of liberty; if we did, *any* task that encountered a trap would end up using the handler written for our task alone. (Besides, we'd have to change not just one, but *all* the vectors in the table – one for each kind of trap.) Instead, we have to use another vector, one that is specific to our task. This is provided in the "Task Control Block", a structure that Exec maintains for each task in the system. A task can get a pointer to its task control block (a "Task" structure) by calling the Exec function "FindTask( )" with a parameter of zero. Once we have the pointer to our task control block, we can change the member called "tc_TrapCode" to point to the code to be executed when an exception occurs.

Changing tc_TrapCode to point to our own routine is very simple. Unfortunately, we can't leave it at that, because our trap code will be executed from within a CPU exception, and trap routines, like interrupt routines, are limited in their capabilities. For example, trap or interrupt code can't call any function that requires multitasking to do its job – this includes "printf( )", commonly used in C programs to print text to the console. Another problem is that during an exception, the stack pointer (A7) points to the supervisor stack, not our task's private user stack, and we don't want to go messing with the supervisor stack (usually). Finally, an exception handler must end with an RTE instruction (ReTurn from Exception), but we probably want to finish execution of our cleanup routine with an 'Exit( )', to remove our process (AmigaDOS's higher-level view of our task) from the system.

As you may have guessed, now comes the fun part where you get to find the solution to these problems. What we have to do is exit from our exception code with an RTE instruction, and *then* have control passed to our special cleanup and exit code. To do that, we have to "mess with the system stack", which, as you may have heard before, you usually don't want to do. In this case, the saved address of the program counter on the stack can be modified so that when the RTE instruction is executed, the CPU will run the program of our choice (the cleanup routine) instead of continuing with the nasty code that caused the exception in the first place. Before the RTE, we have to pull the trap number from the stack, which the system trap handler put there for us. This tells us why the CPU generated the exception – the list of possible trap numbers appears at the end of this article.

A small complication is that the exceptions for "bus error" (trap number 2) and "address error" (trap number 3) push an extra 8 bytes of data onto the supervisor stack. One way to handle this in the trap code is by checking the trap number and advancing the stack pointer past the 8 bytes for trap numbers less than 4.

Another issue that should be addressed by a truly general trap handler is the fact that other members of the 68000 microprocessor family arrange their stacks differently on an exception. Some Amiga users are replacing their 68000s with a 68010 or even a 68020 for added speed. The Amiga's operating system is designed to work with these CPUs, and so should application software, if possible. The trap code could check the CPU type (Exec provides a field for this purpose in the ExecBase structure) and adjust the stack pointer accordingly.

So, to recap the above, the following steps are required to have your task clean up and leave gracefully when a CPU exception occurs:

1) Call FindTask( ) to get a pointer to the task's "Task" structure.
2) Change the "tc_TrapCode" member of the Task structure to point to a short machine language routine that does the following:
   (i) Change the program counter on the stack to point to your "clean up and exit" routine.
   (ii) pull the trap number from the stack
   (iii) perform an RTE instruction to exit the exception handler and pass control to the clean up function.

Your "clean up and exit" function should free any memory your task allocated, close any screens, windows, fonts, libraries, devices, etc. that it opened, and then call "exit( )" (from C) or the equivalent to end and kill the task.

**Our Solution: TrapSnapper**

Lucky for you, all of this has been done for you in the short program presented here called "TrapSnapper". TrapSnapper is set up for C-language trap handling, which takes a bit more set-up than doing it in assembler, since there is some code that *has* to be in assembler.

All you need to do to put a decent trap-handler in your program is:

1) Put a structure template declaration at the start of your file or in a header file that you #include (see listing)
2) Declare an instance of that structure at the top of your file and initialize it with 15 words making up a short machine language routine (see listing).
3) Call the function SetTrap( ) early in the program's execution to initialize the trap handler.

The C code presented here is a simple program that shows how to do this, and demonstrates the effectiveness of the trap handler by generating two kinds of CPU exceptions that would normally result in a Software Error. The program itself just takes the argument supplied on the command line (when the program is invoked from the CLI) and converts it to an integer. If the value is 1, the program forces an address error by attempting to read a word (16 bits) from an odd address. If the value is not 1, the program tries to divide the value into 100 – causing a divide by zero exception, of course, if the value entered was zero.

If this program was compiled without the call to SetTrap( ), it would cause a software error if it was run and given zero or one as an argument. As it stands, with the TrapSnapper in place, it just prints a warning message and the trap number, and exits gracefully, removing itself from the system without a trace.

Here's how the trap code works in C. The initial trap code must be in assembler so that we can access the stack pointer A7 directly and perform an RTE instruction. To do this in C, the machine code is set up as a static array of words and a pointer to this array is put into the Task structure's "tc_TrapCode" member. The machine code needs a pointer to the clean up code (the CleanUpAndExit( ) function in this example), and a place to store the trap number. These long-words are stored immediately before the machine code itself through the use of the "TrapData" structure defined at the beginning of the program. Include this structure template declaration at the top of your file or in a header file that you #include.

The assembly code appears as comments in the C listing for TrapSnapper. The code is fairly straightforward: it pulls the trap number from the stack and stores it in the MyTrap structure where the C code can get at it; adjusts the stack pointer if the trap number was less than 4 to allow for bus and address errors; then replaces the program counter on the stack with the address of the clean-up routine (which was put in the MyTrap structure by the SetTrap( ) function). Finally, it ends with an

RTE instruction. As you may recall, we mentioned that the exception stack frame was CPU–dependent, and a general trap–handler should work for the 68010 and 68020 as well as the 68000. Well, we're good at giving advice, but this trap–handler isn't that general. Sorry, 68010/68020 users, if this routine doesn't work on your machine.

An instance of a TrapData structure (MyTrap) is then declared and initialized – this is where the machine language is created. The pointer to the clean–up code will be put into this structure by the SetTrap( ) function.

The SetTrap( ) function finds the pointer to the task's Task control block, then puts a pointer to the trap machine code into the "tc_TrapCode" member. Finally, it puts a pointer to the function called CleanUpAndExit( ) into the TrapData structure called MyTrap.

The CleanUpAndExit( ) function is where all of the clean–up code for the program goes. In this case, it just prints a message and performs the C function exit( ) (The DOS Exit( ) function could be used instead.) It also prints the exception that was encountered by looking at the TrapNum member of the MyTrap structure. In a more typical program, the CleanUpAndExit function would close any Intuition screens and windows that the program had opened, free any memory it had allocated (including graphics memory like rasters), and close open fonts, devices, and libraries. In short, anything that the program would do to clean up after itself before it exits should be done in CleanUpAndExit( ).

Whether you fully understand the details of the trap handler or not, you can easily add it to your own C or assembler programs. Every program should have its own trap handler to spare the user from the plague of the Software Error when things go wrong. You can't always guarantee that a program is free of bugs, but with TrapSnapper, at least you can make them less harmful.

**List of Trap Numbers:**

**2 – Bus Error**
externally generated signal from Amiga Hardware

**3 – Address Error**
word or longword instruction attempted at odd address

**4 – Illegal Instruction**
a meaningless op–code was encountered

**5 – Division by zero**
the source operand of a DIVS or DIVU instruction was zero

**6 – CHK instruction**
operand of a CHK instruction fell outside of specified bounds

**7 – TRAPV instruction**
overflow (V) set when a TRAPV instruction was executed

**8 – Privilege violation**
a supervisor–state operation was attempted in user state

**9 – Instruction trace**
generated after each instruction while in trace mode

**10 – 1010 Emulator**
an op–code starting with the bit–pattern 1010 was encountered

**11 – 1111 Emulator**
an op–code starting with the bit–pattern 1111 was encountered

**32 through 47 – TRAP instructions**

```
/**                    TrapSnapper From The Transactor                    **
 *
 *  This program shows an easy way to handle CPU exceptions from
 *  a C program. Just include the structs below, and put your error
 *  message and/or cleanup code in the function CleanUpAndExit( ).
 *  call SetTrap( ) to initialize the trap handler. You can get the
 *  trap number as shown in this example.                    (c) 1987 AHA!
**/

#include <exec/tasks.h>

struct Task *MyTask, *FindTask( );

struct TrapData }
    long TrapNum;                   /* trap number will be stored here      */
    int (*Code)( );                 /* pointer to user trap handler function */
    USHORT MLcode[13];              /* trap–handler machine code goes here */
{;

struct TrapData MyTrap = }
    0,
    NULL,    /* clean–up routine address – will be filled in later */
    0x201F,                         /* MOVE.L (A7)+,D0          */
    0x41FA, 0xFFF4,                 /* LEA    *-$A,A0           */
    0x2080,                         /* MOVE.L D0,(A0)           */
    0xB0BC, 0x0000, 0x0003,         /* CMP.L  #3,D0             */
    0x6202,                         /* BHI.S  *+4               */
    0x504F,                         /* ADDQ.W #8,A7             */
    0x2F7A, 0xFFE8, 0X0002,         /* MOVE.L errfunc(PC),2(A7) */
    0x4E73                          /* RTE                      */
{;

/* commented assembler code is below:
 *
 * errnum:  DS.L    1              ;below code will put trap number here
 * errfunc: DS.L    1              ;pointer to clean–up routine goes here
 * start:   MOVE.L  (A7)+,D0       ;pull trap number off stack
 *          LEA     errnum,A0      ;force PC–relative addr mode
 *          MOVE.L  D0,(A0)        ;put trap # in errnum for C
 *          CMP.L   #3,D0          ;check for bus or address error
 *          BHI     other          ;other traps (4 or greater)
 *          ADDQ    #8,A7          ;skip extra info on stack
 * other:   MOVE.L  errfunc(PC),2(A7) ;point PC on stack to user routine
 *          RTE                    ;exit from trap–handler
*/

extern int CleanUpAndExit( ); /* your clean–up function */

main (argc, argv)
int argc;
char *argv[ ];
}
int n1, n2;
int i2, *i1 = 1;

   SetTrap( );  /* set up trap handler */
   if (argc != 2)
   }
       printf("Usage: %s <n>\n", argv[0]);
       exit (0);
   {
   n1 = atoi(argv[1]   *                              *

   /* if arg = 1 let's get an address error by referencing an odd address */
   if (n1 == 1)
       i2 = *i1;

   /* if n1 is zero, we'll get a divide by zero trap */
   n2 = 100 / n1;

   printf("100 divided by %d equals %d\n", n1, n2);
{

SetTrap ( )
/* initialize trap handler */
}
   MyTask = FindTask(0L);
   MyTask->tc_TrapCode = (APTR)MyTrap.MLcode;
   MyTrap.Code = CleanUpAndExit;
{

CleanUpAndExit ( )
/* warn of error, free memory, close up everything, and exit */
}
   printf("Hey – watch it! If I weren't a sophisticated\n");
   printf("Transactor program, I would have bombed out!\n");
   printf("(The problem was, I got a trap number %ld)\n", MyTrap.TrapNum);
   exit(0);
{
```

# Amiga Dispatches

by Tim Grantham, Toronto, Ontario

I'm sure that by now you have all pored over the numerous articles describing the new Amiga 2000. I'm pleased to see that it is pretty much as I described in this column several months ago, with the exception of the lesser amount of RAM. I hope to have an evaluation unit soon, and then I will file my own report. The Amiga 500 (at $649 US) and the 2000 (at $1499 US) are essentially the same machine as the 1000, with the same OS, the same chips, and the same software. Only the packaging is fundamentally different.

What I do find particularly intriguing about the 2000 are slots that apparently will enable one to upgrade the machine to not only faster CPUs like the 68020 and 68881, but also to newer versions of the custom graphics chips. The latter, as I have mentioned in previous columns, will provide access to larger amounts of video RAM and higher resolutions.

Not much is known about the 500, except that it is intended to provide competition for the Atari 1040 and is packaged in a very similar style, with a built–in disk drive, non–detachable keyboard and, for another $150 US, 1 meg. of RAM and a real–time clock/calendar.

Where does this leave us 1000 owners? Commodore has said that they will continue to make the 1000 as long as there is a demand for it. Some fear this means we shall never see another. However, Commodore has been saying the same thing about the 64 for several years now, and that machine is still with us.

Software will not be a problem, for reasons mentioned above.

The only area of concern is hardware. Will third–party peripheral providers abandon the 1000's 86–pin expansion bus for the Zorro slots of the 2000? Certainly, Zorro cards are easier to design than the external units needed by the 1000. No housing is required, no power supply is required, no worry about FCC and CSA RFI standards is necessary. But there are a couple of factors in the 1000's favour.

First, Zorro card makers will be competing with Commodore; makers of expansion units for the 1000 will not. Second, the 500 will have almost the same 86–pin expansion slot as the 1000: the differences would involve only minor changes to 1000 expansion products – an incentive for manufacturers to maintain their 1000 product lines. Finally, there are still 150,000 1000's out there – not a bad market. I think that, after some shake–out, we will see plenty of products for the 1000 still available. Furthermore, I believe we will see expansion boxes for both the 500 and 1000 that will let one plug in Zorro cards, and perhaps even the so–called Bridge and PC cards, designed for the 2000.

I also believe there will be continuing demand for the 1000, though obviously not as much as there has been. The detachable keyboard and internal power supply are better than their counterparts on the 500. And peripherals like the Genlock and Digiview are currently only available for the 1000. Eventually the 1000 will fade out of the picture, but certainly not in the same fashion as such orphans as the PC Jr. or the 128K Macintosh.

## Software News

Dropping around at an Amiga dealer's store, I was surprised again by the quantity and quality of the software available for the Amiga. But even more will come along as the 500 opens up the home market, and the 2000 finds a niche for itself in the business and scientific world.

At the Club–Amiga meeting here in Toronto, John Skeel, Vice–President of marketing for Aegis Development, demoed their latest products **Sonix, Draw Plus, Aegis Animator II**, and **Diga!**. This last one is a terminal program using a proprietary protocol that permits file transfers between two Amigas (each running Diga!) while remaining in CHAT mode.

He also showed an extraordinary video tape of a 3D animated sequence, every frame of which had been created on a 512K Amiga 1000 using their new software. The program can draw a frame in less than ten seconds, depending on the complexity of the image. It can also, using an optional frame controller, record each image on a VCR. The program uses overscan so that the picture is not confined within the borders of the usual Amiga screen, and can use fractal geometry to generate landscapes.

The artist describes the sequence in a text file. She can then test the sequence in a preview animation mode that uses wire–frame models. Once she is happy with the sequence, the solid three–dimensional, 16–colour output is generated automatically. Skeel said that the demo tape, which lasted about a minute, took 8 hours to generate from previously completed scripts. By my estimation, that means that Amiga generated and recorded at least 1800 frames in 8 hours. I wonder how many it could have done with the 68020/68881 combination installed? The program is supposed to be available as you read this, for a cost of about $300 US.

**Deluxe Paint II** from Electronic Arts is a significant upgrade to the original **DPaint**. Besides being able to handle multiple resolutions, it adds a unique ability to rotate a brush through three dimensions. . . According to DJJAMES, a sysop on People-Link's Amiga Zone, the **OpenLibrary()** function in 1.1 didn't care what library version number you passed to it. Another version of **OpenLibrary()** was created in 1.2 that fixed that bug and it's called **NewOpenLibrary()**. . . Speaking of 1.2, mutual exclusion of gadgets is *still* not implemented. R.J. Mical suggests a roll–your–own approach that entails removing the gadgets involved, making the state changes in the gadget structures, and then re–attaching the gadgets, using the 1.2 functions **RemoveGList(), AddGList()** and **RefreshGList()**. . . Larry Phillips of Vancouver, mentioned two issues ago as a sysop on The Source, is now a sysop on CompuServe's Amigaforum. He's a valuable addition to a good team. . .

I downloaded some terrific Public Domain software and Shareware recently: **Blitzfonts**, by a gentleman named Hayes C. Haugen, speeds up screen text output on the Amiga 2–3 times. I have used it with Uedit and Scribble! to great effect. It's written entirely in ML and uses up less than 4K of memory once installed. Mr. Haugen is asking only a $10 donation. If you've got it, send it. Another fine program being offered as Shareware is Dave Wecker's ray–tracing software. This program can generate stunning 320 by 400 HAM graphics. A PD assembler and C compiler have also arrived on the scene. This is good news but I anticipate a problem here. Both of these require the Amiga include files to work. Those, however, are copyrighted by Commodore and licensed to people like Lattice, Manx and Metacomco. Wouldn't it be nice of Commodore to offer these includes to the general public, say for a nominal copying fee? Are you listening, Commodore?

Speaking of languages, Manx has released version 3.4 of the **Aztec C** compiler by Jim Goodnow. In addition to further optimization of the compiler, it now fully supports scatter–loading, 1.2 and the 68020/68881 processors. The prices have also been significantly lowered: $199 for the personal version, $299 for the developer's version and $399 for the commercial version. TDI has also released a new version of their Modula–2 compiler, version 3.00. . . Ben Blish's SoftCircuits company has lowered the price of their **PCLO** (Printed Circuit Board Layout) software to $499. A greatly enhanced version called **PCLO Plus** is now selling for $1024. All owners of the original **PCLO** will have their copies updated to **PCLO Plus** for a nominal charge. . .

Charlie Heath of BIX and MicroSmiths is heading up a group of volunteers who want to replace all the BCPL–encoded components of AmigaDOS with equivalents written in assembly language. A number of the commands have already been done. Meanwhile, rumour has it that Tim King of Metacomco, the original programmer of AmigaDOS, is working on optimizations that will permit the efficient use of hard disk drives, and the folks at the Software Distillery are also working on improving AmigaDOS. As I understand it, these involve no change in Kickstart, only in Workbench. . . **TEX**, the sophisticated typesetting system developed by Donald Knuth of Stanford University, is available for the Amiga from N² Computer Consultants, P.O. Box 2736, College Station, TX 77841. . .

Finally. a word about **Transformer**. Those of us who use it know by now that it will not work under 1.2. It apparently uses a bug in 1.1 to load that was fixed in 1.2. Commodore now has in their hands a 1.2 version of **Transformer** and will hopefully upgrade us **Transformer** 1.1 owners ASAP. Do you remember Commodore's promises to provide a hardware Accelerator that would speed up operation of the **Transformer** to something approaching full speed? Well, apparently somebody in marketing had the bright idea that making a full–sized clone one stuck on the side would make them a lot more money. Thus the SideCar was born, and the Accelerator accelerated to an early grave. Meanwhile, Simile, the company that wrote the **Transformer**, was told to take a hike. CBM will not allow them, apparently, to sell a version of the software that would offer colour and graphics. This is one of the few Amiga products that Commodore has reneged on (though some were a long time coming) and we and Simile certainly deserve better treatment.

## Hardware News

There are two hardware products for the Amiga worthy of special mention: the Insider internal 1 meg. RAM board; and Byte–by–Byte's PAL Jr. expansion box.

The Insider mounts inside the Amiga and attaches directly to the 68000 bus. In addition, it is configured to lie at the $C0000000 address reserved under 1.2. It is true FAST RAM with no wait states, unlike some other internally–mounted RAM expansions that piggyback on to the video memory chips. Because it lies in the reserved area of memory, a full 8 megabytes can still be added on the expansion bus, bringing the total possible amount of memory on the Amiga to 9.5 Megabytes. The unit is one of the few memory boards that will work with the Sidecar.

Some have expressed concern that it may prevent other units on the expansion bus from working because the Amiga bus is rated only for one 'F' load, which would be absorbed by the Insider. However, the manufacturers say that they have buffered all except the data lines. Larry Phillips has one installed in his machine and says he has had no trouble driving an external memory board or the Sidecar. He feels there is probably enough leeway in the specs to allow units on the expansion bus to coexist with the Insider. I must admit, it would be very nice to have a 1.5 meg. Amiga with a battery–backed clock/calendar that doesn't have all that junk hanging off the side. It is available

from Michigan Software, 43345 Grand River, NOVI, MI 48050 at a cost of $349.95. The telephone number is 313–348–4477.

Byte–by–Byte have announced that they have completely redesigned their PAL Jr. expansion box. It now sits on top of the Amiga 1000, is a little under three inches high, and provides a 20 meg. ST506 hard disk with a DMA controller, a board with 1 meg. of RAM, and an empty slot for another Zorro board. It provides its own power with a 90–watt supply on board. Buyers can upgrade the controller board to include a SCSI controller as well. The supplied RAM, like the Insider, resides at the $C0000000 address. A clock is also standard equipment. The unit costs $1495 US and can only be ordered directly from Byte–by–Byte.

As a little postscript, it is interesting to note that the custom chip each manufacturer inserts into their expansion product to enable it to autoconfigure, also contains a product code and a manufacturer's ID#. Thus, software can check to see exactly what hardware is on the bus, and adapt accordingly, without the user having to go through lengthy installation procedures.

**Report from a Sunday programmer**

I'd like to finish up this edition of Amiga Dispatches with a recounting of my Amiga programming efforts (so you can stop here if it's only product news you want to read about). Not that I'm any kind of programming avatar – quite the opposite. But my experiences may provide insight for some and amusement for others.

In case you don't know, I ascended from the lowly depths of the 65xx world to the lofty heights of the Amiga. I only just recently, and reluctantly, sold my C64 system. I still view the 64 as a very fine computer: I don't think there is a better machine for anyone who wants to learn the basics. It helped bring about a revolution in the way the average citizen perceived computers: not as modern metal Minotaurs tended by priests in white shirts and black ties but rather as extensions of themselves – brain transportation, if you'll permit me.

But back to the Amiga.

It's been a year now and I still get a gee–whiz kick out of it. But I wasn't content to be only a user. I wanted to see what *I* could make it do. I felt reasonably confident. After all, I could find my way around the 64 in BASIC and 6510 machine language. The Amiga would simply be the next step up the ladder. Fools rush in. . .

I did not even try to learn AmigaBASIC. While it was apparent that it was a much more powerful BASIC than any on other CBM machines, its editor was clumsy – a shame, considering that the Amiga was supposed to be a friendly machine. Furthermore, if you wanted to multitask with other AmigaBASIC programs, you had to run other copies of AmigaBASIC. Finally, while it was possible to call the system routines from AmigaBASIC, it was hardly the ideal way of using them. It was clear that this implementation of BASIC had been stuck onto the Amiga like training wheels on a Harley–Davidson. All right, so I exaggerate a little – more like a cow–catcher on Porsche.

The reality was that almost all software development was being done in C, a language I had only recently heard of, and the rest in 680xx machine language. Most of the operating system had been written in C and the documentation from Commodore assumed a C environment. I felt that it was perhaps propitious that the Amiga was making me seriously consider learning this language: C *is* supposed to be the language of the eighties. What was true of the Amiga was increasingly true for microcomputers in general: C provided a language that significantly reduced development time while providing most of the speed of machine language. So I obtained a copy of Abacus' **Super–C** for the 64, and the well–known *C Primer Plus*, published by The Waite Group, and began.

It wasn't easy. Forget the articles that try to use BASIC as a springboard to C – they are simply too different. First and foremost, C is a *structured* language. This isn't just another buzzword. If you have come from a linear programming environment, like CBM BASIC and ML, you'll run smack into a completely different way of thinking. I think I must have added new cortical folds to my brain, trying to twist my linear thought processes around to accommodate the modularity of C. I can't tell you how often I longed to use a **goto**, especially in loops. (**goto** is implemented in C, but using it is considered slumming by purists. I decided that, just for the mental discipline, I would also avoid using it.)

Furthermore, C is emphatically not a friendly language. It was designed by programmers for programmers. As such, it is a mid–level language that combines the sophisticated data structure handling capabilities and modularity of a high level language with the efficiency and flexibility of assembly language. If you have some experience with assembly language, you will indeed feel more at home. But you had better comment your source code heavily if you want anybody, including yourself, to understand what you've written six months down the road.

Is it worth the trouble? Absolutely. Once you've wrenched your brain around to C's way of thinking, you'll feel right at home programming the Amiga. C was written in conjunction with, and as an aid to, the development of Unix. The Amiga OS has many things in common with Unix – multitasking, message ports, system structures – and C is becoming the lingua franca of multitasking operating systems. Not to mention the fact that nearly all the examples given in the official Amiga docs are in C.

Assembly language is not out of the question on the Amiga. 680xx ML is much more powerful and orthogonal than 65xx ML, but they both have linearly addressed memory and similar opcodes. The jump from 65xx to 680xx will not leave your head spinning. And it has the advantage of producing smaller, tighter code than C. Indeed, the Manx Aztec C compiler for the Amiga actually produces assembly language on its first pass, which it then assembles on the second pass. You can examine the assembly language object code if you wish. If you wish to learn 680xx machine language, I can recommend the *68000, 68010,*

*68020 Primer* by Stan Kelly–Bootle and Bob Fowler, also published by The Waite Group.

There are many other languages for the Amiga but few provide the tight interface to the Amiga libraries that C does.

Learning C, though, had only brought me half way. There was still the multitasking, multiprocessing Amiga OS to learn.

Bear with me a moment while I conjure up the following scenario: You sit behind a desk in a room. In that room are a giant screen, a pair of speakers, one of those pneumatic systems that lets you send and receive office memos in little cylinders, and a locked door with a mail slot. (I leave it to you to add whatever companions or libations you need to feel at home.) The screen is black, containing only the words 'Guru Meditation' in red letters. No sound issues from the speakers.

All around you, in the floors above and below, and outside that locked door, you can sense an enormous amount of activity. Who knows how many other rooms there are, with other individuals behind other desks, busily receiving, processing and sending memos and forms. And why?

Your head snaps up, for the image on the screen has changed. Now it is filled with a familiar picture of words dropping down from a bar at the top of the screen, and one of the words is highlighted. Suddenly, with a hollow THWONK!, a cylinder pops out of the incoming message tube. You open it. Inside is a piece of paper, a small form actually, labelled 'IntuiMessage' at the top, and someone, or something, has filled in the various blanks below. One jumps out at you: in the blank labelled 'Class', the word 'MENUPICK' is printed.

Now you know what it is you must do. You make a copy of the IntuiMessage for your records, initial it, mark 'Return to Sender' on it, and stick it into the outgoing message tube. With a muffled THUPP!, it disappears. Quickly, you fill out a directive labelled 'AutoRequest'. You know that, as long as you supply every required detail, and submit every required form, this hyper–efficient bureaucracy will follow your instructions to the letter.

You hurry over to the door and pop the directive through the slot. Even as you turn away, another image appears on the screen, a rectangle containing the message 'Are you sure?' and two smaller rectangles that say 'OK' and 'Cancel'. Then the screen goes black once more, 'Guru Meditation' appears in red letters, and you know that soon another cylinder will pop, THWONK!, out of the incoming message tube.

As you head back to your desk, to spend your allotted time filling out forms, receiving forms, processing messages, and sending directives, you wonder who or what is outside that locked door. For you sense that, if you knew how, you might be able to fill your room with ever more complex images and sounds, or control the images and sounds in someone else's room.

This scenario, fanciful though it is, is very much like what programming in the Amiga's multitasking environment is like. Even some of the terminology is the same: one 'submits' an initialized structure (completed form); one 'processes' messages received in the MsgPort (message tubes) from MsgPorts in other tasks (rooms). Only the Amiga OS is infinitely more efficient and integrated than the most highly automated, networked and integrated office system could ever be.

Working in that single room is analogous to a programmer who simply wants to use the interface that Intuition provides for his or her program. It's not necessary to know what sends those messages, for example, only what response is required to fulfill the programmer's objectives. If the programmer needs to know or control what is going on in other rooms – tasks, I mean – they must open that locked door and go exploring.

A program that lets you do just that is the **Structure Browser** program written by Chris Zamara and Nick Sullivan and presented in last month's *Transactor*. Using **SB** is almost like playing one of those text adventures where you explore the rooms of a huge castle, and find out where all the secret tunnels are that link one room with another. Only it's better than that because the Amiga's display, in the case of the Intuition structures, provides a map of the journey.

Using **SB** proved to be a major watershed in my understanding of the Amiga OS. I think it would be the same for anyone wishing to gain an intuitive grasp of how the various system structures (there are nearly 100 of them) are connected to each other, and how the display reflects their contents. The current version of **SB** supports only the Intuition structures. I hope others will add the other libraries.

Other very helpful utilities for programming on the Amiga include **egad**, the PD gadget editor produced by John Draper et al; **Uedit**, a very fine programmable Shareware text editor written by Rick Stiles; and Charlie Heath's **getfile** file requester. I have used all of these in my first major effort, **Keep v1.0**, which hopefully will be available on the first *Transactor* Amiga disk. **Keep** is a program that is descended from **excise**, a program written in BASIC and ML for the 64 and the PET 8032 by Nick Sullivan. It lets one extract selected messages from a file of messages downloaded from one of a number of information services – simple but useful.

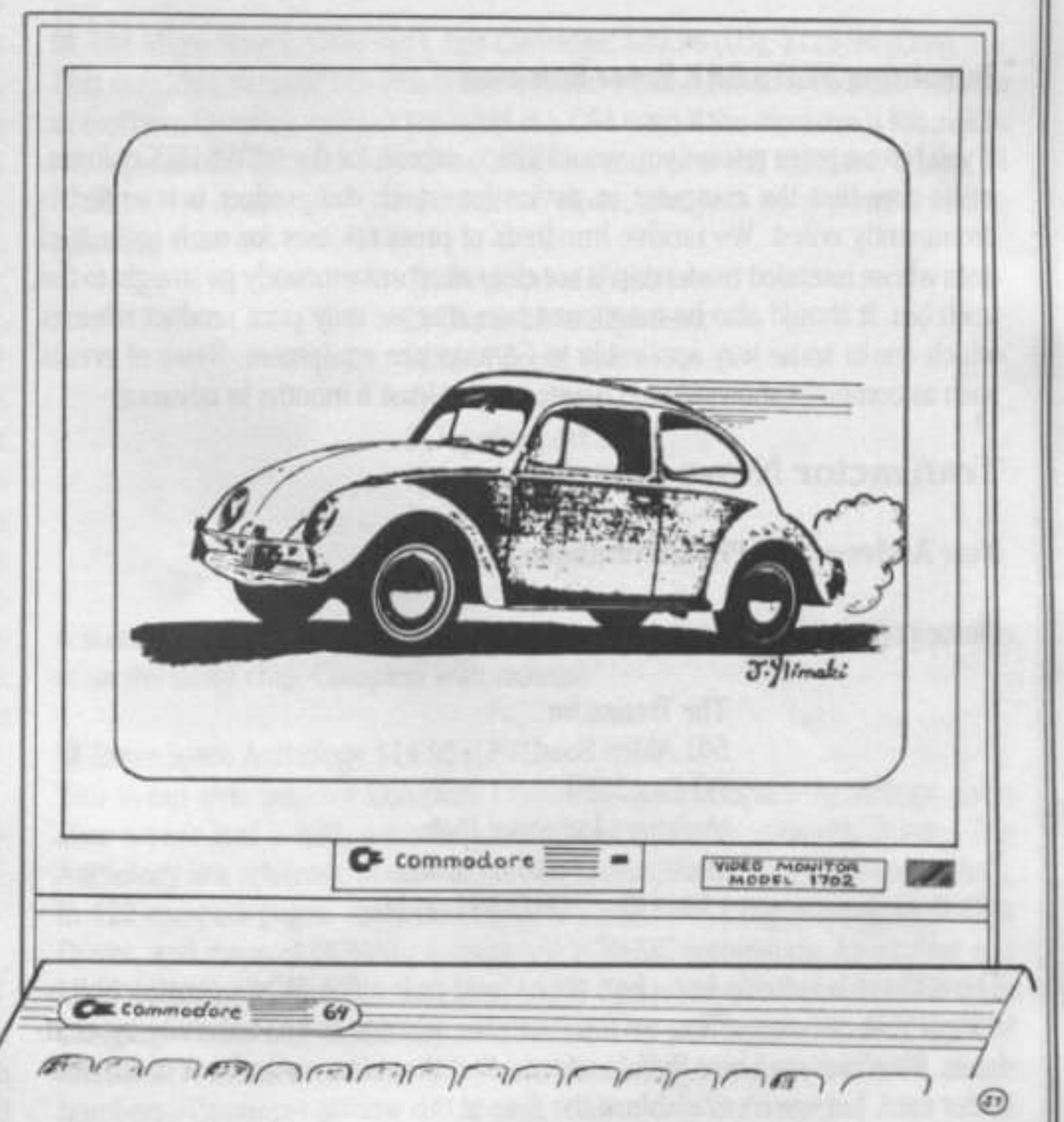
Some of the books I found useful in my programmer's progress were the Amiga Programmer's Guide from Compute! Publications; the Amiga Programmer's Handbook by Eugene Mortimer, published by Sybex; and of course, the best of them all, the Amiga Intuition Reference Manual by R.J. Mical and Susan Deyl, published by Addison–Wesley. The Sybex book is noteworthy because, to my knowledge, it and the new edition of Bantam Books AmigaDOS Reference Guide are the only books that provide information about 1.2.

Before I bid you adieu, let me remind you that any comments or questions about this column, can be sent to me on PeopleLink (AMTAG) or on Compuserve (71426,1626). I can't promise I'll reply but I will certainly do my best.

exit(1);

# Computoons

"He's heavily into computer animation. . ."




DRAT IT! ANOTHER BUG IN MY PROGRAM

"HEY, GUYS! LOOK WHAT THE WIZARD GAVE ME!"

# News BRK

**Submitting NEWS BRK Press Releases**

If you have a press release you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way applicable to Commodore equipment. News of events such as computer shows should be received at least 6 months in advance.

## Transactor News

**New Address and Phone Number**

Please note that The Transactor now has a new address:

The Transactor
501 Alden Road
PO Box 3250
Markham Industrial Park
Markham, Ontario, Canada
L3R 6G6 (416) 737-2786

The address is actually just a box at our local post office. When construction is complete at our new office, we'll publish the address there, hopefully by next issue. We also have a new Buffalo address that should be on the back of our mail order card, but wasn't available at the time of this writing.

**Subscription Intersection Set**

On closer inspection we have found the overlap of TPUG members and Transactor subscribers to be about 1000, not 400 as previously reported. During our first comparison, many went undetected due to differences between the two mail list databases. It seems that a computer just can't tell if two names are the same when a middle initial is included in one but not the other. To compensate, a program was written that bordered on artificial intelligence. However, it's still possible that some matches were missed, especially if the postal/zip code was different in the two databases. So, if you're a TPUG member AND a Transactor subscriber, and you're still getting two issues, please let us know and the two will be combined.

The matches that were found, plus any that we've been informed of by mail, have all been dealt with as of this issue. Any Transactor subscribers that were also TPUG members have had their TPUG memberships extended by the number of issues remaining in their Transactor subscriptions. If you fall into this "intersection set", your mailing label should show this extension.

Since you'll now only be receiving magazines from TPUG, your renewal notice will also come from TPUG. Naturally you should renew to one OR the other. Renewing to both will mean you'll start getting two magazines all over again, and after we get this situation sorted out, we'd like it to be eliminated for good. The fact is, we could make a career outa this. So, since all the existing overlapping memberships/subscriptions have been combined and extended, any and all new ones will get two magazines; one with an insert from TPUG, and another without an insert from us. This applies mainly to those who are currentiy Transactor subscribers and are *not* TPUG members, but considering becoming one.

If you're renewing a subscription, or subscribing for the first time, you'll have the choice of getting the regular magazine (no TPUG insert) or becoming/remaining a TPUG member, and getting the 8-page TPUG insert as part of your Transactor. If you want just the Transactor, use the insert card in the magazine to subscribe/renew. If you want the insert as well, send your TPUG renewal form to TPUG (they'll send it to you before your current membership expires) or, if you're not currently a member, contact them about becoming one. Please do NOT use the postage paid Transactor subscription card for TPUG memberships. TPUG and Transactor are at seperate addresses, and applying to one via the other will delay processing unecessarily.

The current cost of a TPUG 'associate' membership is $25.00 (US/C). For that you get the insert plus access to TPUG's large public domain disk library, neither of which comes with the $15.00 (US/C) subscription to the Transactor.

**Disk Subscription Notes**

Many of those who fall into the intersection set described above also have Transactor Disk subscriptions. These will still be handled by us, and when they expire, a notice will be sent. Some have pointed out that, "we would find it mush easier to renew to both if they were to both expire at the same time. Now that our memberships/subscriptions have been combined, the magazines end at one issue and the disks at another". Good point. What we suggest is this: when you renew your disk subscription, add $7.50 (US/C) for every disk necessary to make your disk and magazine subscriptions concurrent.

**Free Transactor T's with Mag + Disk Subscription**

Subscribe or renew to a combination magazine and disk subscription, and we'll send you a free Transactor T-Shirt! You save 29% off the magazines, 16% off the disks, and get a Transactor T worth $13.95 ($17.95 if you order the jumbo size!) The T-Shirts come in 5 sizes (red only), with a 3-color screen featuring Duke, our mascot, dressed in a snappy white tux, standing behind the Transactor logo done in yellow with black "3-D" borders. The screen was done using a special "super-opaquing" process that cost us quite a bit more than those decals that crack and fade. Mine has been through the wash at least 25 times now, and it still shows virtually no sign of wear due to "washing machine punishment".

**Subscriber Mail Orders**

If you're a Transactor subscriber, and you're using the postage paid order card to purchase items other than a subscription, please write your subscriber number on the card. This way your order is recorded along with your subscription information in our database.

**Customs/Duty on Hardware Products**

Shipping hardware to the US from Canada often incurs customs and/or duty charges at the destination. Some of our suppliers are in the US and for US orders processed by us, we have the items shipped direct without bringing them into Canada. However, other hardware items manufactured in Canada and sent to US destinations may arrive with a surcharge payable. The Transactor cannot be responsible for these charges. This may also add to delivery delays. If you've placed an order for a hardware item and it seems to be taking a rather long time to arrive, it may be sitting at your local customs office, in which case a notice is probably on its way for you to come pick it up.

**Transactor Mail Order**

The following details are for products listed on the mail order card. If you have a particular question about an item that isn't answered here, please write or call. We'll get back to you and most likely incorporate the answer into future editions of these descriptions so that others might benefit from your enquiry.

■ Moving Pictures - the C-64 Animation System, $29.95 (US/C)
This package is a fast, smooth, full-screen animator for the Commodore 64, written by AHA! (Acme Heuristic Applications!). With Moving Pictures you use your favourite graphics tool to draw the frames of your movie, then show it at full

animation speed with a single command. Movie 'scripts' written in BASIC can use the Moving Pictures command set to provide complete control of animated creations. BASIC is still available for editing scripts or executing programs even while a movie is being displayed. Animation sequences can easily be added to BASIC programs. Moving Pictures features include: split screen operation – part graphics, part text – even while a movie is running; repeat, stop at any frame, change position and colours, vary display speed, etc; hold several movies in memory and switch instantly from one movie to another; instant, on-line help available at the touch of a key; no copy protection used on disk.

■ Volksmodem 12, w/cable, and CIS Intro-Pack, $329.00 (Cdn), $199 (US)
Not only do you get the Volksmodem 12 (DOC approved), but you get the cable at no extra charge (the C64 cable goes directly onto the User Port, and the RS232 cable is for any standard RS232 DB-25 female connector) Plus you'll receive a free CompuServe Intro-Pak which contains a User ID, a Password, and $15.00 of connect time! The Volksmodem 12 will work at 300 or 1200 baud, and is "Hayes compatible" so it will work with virtually any terminal software because the commands are controlled by you from the keyboard – just type "AT" (for ATtention) and follow with any of several easy-to-remember commands – no special POKing or elaborate dialing routines necessary! (I've been using a Hayes for almost 3 years, and my Volks for over a year - I love them both! - KJH) It comes with (get this) a 5 year manufacturer's warranty on parts and labour! The modem is shipped insured via UPS at no extra charge.

■ Intelligent I/O Interface Cards
■ BH100 I/O Interface Card w/documentation $129 (US), $199 (Cdn)
■ BH100–AD8 8-Channel A to D Conversion Module $45 (US), $69 (Cdn)
■ BH100 Beginners Course $159 (US), $239 (Cdn)
■ BH100–S Security System $25 (US), $39 (Cdn)
These products from Intelligent I/O will make great Christmas gifts! And if you've been wondering what to do with that VIC 20 that doesn't get much attention anymore, they're perfect! If you've ever wanted to start doing some real world interfacing, real easy, and inexpensively, then these items are ideal. The boards they sent us for evaluation are currently watching for floods in my basement. Too bad I didn't think of it before the flood – it only took about an hour using spare parts I had lying around – no resistors, no capacitors, just two strips of metal, a piece of styrofoam, a brick, and about 20 feet of wire that was also collecting dust. Once I get time, I intend to make it do some more surveillance since only one channel is currently in use. And the program to do it? A quick and messy 5 lines! Since the boards are memory mapped through the cartridge port, a PEEK is all you need! The 22 page manual is clear and concise. All products come with a 90 day manufacturer's warranty. Shipped insured via UPS at no extra charge.

■ Transactor T-Shirts, $13.95 and $17.95 (US/C)
As mentioned earlier, they come in Small, Medium, Large, Extra Large, and Jumbo. They're 13.95 each, $17.95 for the Jumbo. The Jumbo makes a good night-shirt/beach-top – it's BIG. I'm 6 foot tall, and weigh in at a slim 150 pounds – the Small fits me tight, but that's how I like them. If you don't, we suggest you order them 1 size over what you usually buy. The design is screened using a "super-opaquing" process so they wear much longer than your ordinary screens and iron-ons.

■ The Transactor Book of Bits and Pieces #1, $14.95 (US/C)
Not counting the Table of Contents, the Index, and title pages, it's 246 pages of Bits and Pieces from issues of The Transactor, Volumes 4 through 6. Even if you have all those issues, it makes a handy reference – no more flipping through magazines for that one bit that you just know is somewhere. . . Also, each item is forward/reverse referenced. Occassionally the items in the Bits column appeared as updates to previous bits. Bits that were similar in nature are also cross-referenced. And the index makes it even easier to find those quick facts that eliminate a lot of wheel re-inventing.

■ The Tr@ns@ctor 1541 ROM Upgrades, $59.95 (US/C)
You can burn your own using the ROM dump file on Transactor Disk #13, or you can get a set from us. There are 2 ROMs per set, and they fix not only the SAVE@ bug, but a number of other bugs too (as described in P.A. Slaymaker's article, Vol 7, Issue 02). Remember, if SAVE@ is about to fail on you, then Scratch and Save may just clobber you too. This hasn't been proven 100%, but these ROMs will eliminate any possibilities short of deliberately causing them (ie. allocating or opening direct access buffers before the Save).

■ The Micro Sleuth: C64/1541 Test Cartridge, $89.95 (US), $129.95 (Cdn)
This cartridge, designed by Brian Steele (a service technician for several schools in southern Ontario), will test the RAM of a C64 even if the machine is too sick to run a program! The cartridge takes complete control of the machine. It tests all RAM in one mode, all ROM in another mode, and puts up a menu with the following choices:

1) Check drive speed
2) Check drive alignment
3) 1541 Serial test
4) C64 serial test
5) Joystick port 1 test
6) Joystick port 2 test
7) Cassette port test
8) User port test

A second board, that plugs onto the User Port, contains 8 LEDs that lets you zero in on the faulty chip. Complete with manual.

■ Inner Space Anthology $14.95 (US/C)
This is our ever popular Complete Commodore Inner Space Anthology. Even after a year and a half, we still get inquiries about its contents. Briefly, The Anthology is a reference book – it has no "reading" material (ie. "paragraphs"). In 122 compact pages, there are memory maps for 5 CBM computers, 3 Disk Drives, and maps of COMAL; summaries of BASIC commands, Assembler and MLM commands, and Wordprocessor and Spreadsheet commands. Machine Language codes and modes are summarized, as well as entry points to ROM routines. There are sections on Music, Graphics, Network and BBS phone numbers, Computer Clubs, Hardware, unit-to-unit conversions, plus much more. . . about 2.5 million characters total!

■ AX1000 Amiga 1 MEG RAM Box $729.00 (+$100 S&H) (US),
$1035.00 (+$25 S&H) (Cdn)
■ AX2000 Amiga 2 MEG RAM Box $899.00 (+$100 S&H) (US),
$1276.00 (+$25 S&H) (Cdn)
The AX2000 adds 2 Megabytes of "fast" RAM to the Amiga, allowing more tasks to run in the system at once, or for use as a fast RAM-drive. The unit plugs into the expansion connector on the side of the Amiga and duplicates the connector for other devices to plug into. Up to two RAM boards may be plugged in together (limited by the Amiga'a power supply), adding 4 Megabytes. The box has "auto-config", so with Kickstart 1.2 the RAM will automatically be added to the system when it is booted. If you are using Kickstart 1.0 or 1.1 (no auto-config), you can use the program included with the AX2000 to add the memory to the system, and change your startup-sequence to automatically add the memory on power-up. Standard expansion bus architecture was used in the design of the AX2000, ensuring compatability with all peripherals and operating system releases. The unobtrusive steel box is the same height and colour as the Amiga, and snugs up to the side without taking up much extra space. The unit is built tough and comes with a 1 year manufacturer warranty.

This seems to be the most highly-recommended Amiga RAM board, and the first one to actually be available, so we're selling it here at The Transactor. You can order the AX2000 or the 1-Meg AX1000 from the subscription form in this issue. Shipping and Handling to the USA. is via courrier and includes all customs clearance, or you can opt to clear shipments yourself and have it shipped "collect".

| | | |
|---|---|---|
| ■ Superpak 1.0 | C64 | $49.95 (US), $59.95 (Cdn) |
| ■ Pocket Writer | C64 | $29.95 (US), $39.95 (Cdn) |
| ■ Pocket Planner | C64 | $29.95 (US), $39.95 (Cdn) |
| ■ Pocket Filer | C64 | $29.95 (US), $39.95 (Cdn) |
| ■ Superpak 1.0 | C128 | $59.95 (US), $69.95 (Cdn) |
| ■ Pocket Writer | C128 | $39.95 (US), $49.95 (Cdn) |
| ■ Pocket Planner | C128 | $39.95 (US), $49.95 (Cdn) |
| ■ Pocket Filer | C128 | $39.95 (US), $49.95 (Cdn) |

■ Pocket Dictionary $14.95 (US), $19.95 (Cdn)
Version 2.0 of the software trio from Digital Solutions is now in production. The new packages include both the 64 and 128 versions on the same disk. Each 2.0 Pocket package will sell for $59.95 (US). or $84.95 (Cdn). A Superpak will include all three for $99.95 (US) or $139.95 (Cdn). The Pocket Dictionary is still $14.95 (US), $19.95 (Cdn). However, they won't be available from us until next issue.

Version 1.0 is still available, and at terrific prices! The 64 and 128 versions still come in separate packages, but the real deal is the special price for all three. The C64 Superpak is $49.95 (US) or $59.95 (Cdn). C128 Superpaks are $59.95 (US) or $69.95 (Cdn). To top it off, we'll throw in the Pocket Dictionary program for free! If you average the price of all four, it comes to less than the price of two!

■ The TransBASIC Disk $9.95 (US/C)
This is the complete collection of every TransBASIC module ever published up to Volume 7, Issue 01. There are over 120 commands at your disposal. You pick the ones you want to use, and in any combination! It's so simple that a summary of instructions fits right on the disk label. The manual describes each of the commands, plus how to write your own commands.

■ Super Kit 1541 $29.95 (US), $39.95 (Cdn)
Super Kit is, quite simply, the best disk file utility there is. No more losing those valuable copy-protected originals (like what's happened to me twice too many times). So far we've shipped over 600 Super Kits and orders continue to pour in.

■ Gnome Speed Compiler $59.95 (US), $69.95 (Cdn)
This compiler is for BASIC 7.0 on the Commodore 128.

■ Gnome Kit Utility $39.95 (US), $49.95 (Cdn)
Gnome Kit is a Commodore 128 utility with enhancements for the BASIC editor (like Trace, Find, Renumber, Delete, Auto, etc.) as well as enhanced monitor commands, and floppy disk monitor functions.

### Transactor Disks, Transactor Back Issues, and Microfiche

All issues of The Transactor from Volume 4 Issue 01 forward are now available on microfiche. According to Computrex, our fiche manufacturer, the strips are the "popular 98 page size", so they should be compatible with every fiche reader. Some issue are ONLY available on microfiche – these are marked "MF only". The other issues are available in both paper and fiche. Don't check both boxes for these unless you want both the paper version AND the microfiche slice for the same issue.

To keep things simple, the price of Transactor Microfiche is the same as magazines, with one exception. A single back issue will be $4.50 (US/C) and subscriptions are $15.00 (US/C). The exception? A complete set of 18 (Volumes 4, 5, and 6) will cost just $39.95 (US/C)!

This list also shows the "themes" of each issue. "Theme issues" didn't start until Volume 5, Issue 01. The Transactor Disk #1 contains all program from Volume 4, and Disk #2 contains all programs from Volume 5, Issues 1–3. Afterwards there is a separate disk for each issue. Disk 8 from The Languages Issue contains COMAL 0.14, a soft-loaded, slightly scaled down version of the COMAL 2.0 cartridge. And Volume 6, Issue 05 published the directories for Transactor Disks 1 to 9.

■ Vol. 4, Issue 01 (■ Disk 1)
■ Vol. 4, Issue 02 (■ Disk 1)
■ Vol. 4, Issue 03 (■ Disk 1)
■ Vol. 4, Issue 04 – MF only (■ Disk 1)
■ Vol. 4, Issue 05 – MF only (■ Disk 1)
■ Vol. 4, Issue 06 – MF only (■ Disk 1)
■ Vol. 5, Issue 01 – Sound and Graphics (■ Disk 2)
■ Vol. 5, Issue 02 – Transition to Machine Language – MF only (■ Disk 2)
■ Vol. 5, Issue 03 – Piracy and Protection – MF only (■ Disk 2)
■ Vol. 5, Issue 04 – Business & Education – MF only (■ Disk 3)
■ Vol. 5, Issue 05 – Hardware & Peripherals (■ Disk 4)
■ Vol. 5, Issue 06 – Aids & Utilities (■ Disk 5)
■ Vol. 6, Issue 01 – More Aids & Utilities (■ Disk 6)
■ Vol. 6, Issue 02 – Networking & Communications (■ Disk 7)
■ Vol. 6, Issue 03 – The Languages (■ Disk 8)
■ Vol. 6, Issue 04 – Implementing The Sciences (■ Disk 9)
■ Vol. 6, Issue 05 – Hardware & Software Interfacing (■ Disk 10)
■ Vol. 6, Issue 06 – Real Life Applications (■ Disk 11)
■ Vol. 7, Issue 01 – ROM / Kernel Routines (■ Disk 12)
■ Vol. 7, Issue 02 – Games From The Inside Out (■ Disk 13)
■ Vol. 7, Issue 03 – Programming The Chips (■ Disk 14)
■ Vol. 7, Issue 04 – Gizmos and Gadgets (■ Disk 15)
■ Vol. 7, Issue 05 – Languages II (■ Disk 16)
■ Vol. 7, Issue 06 – Simulations and Modelling (■ Disk 17)
■ Vol. 8, Issue 01 – Mathematics (■ Disk 18)

## Industry News

The following items, compiled by Astrid Kumas, are based on press releases recently received from the manufacturers. Please note that product descriptions are not the result of evaluation by The Transactor.

### National Computer Conference 1987

This announcement is to remind all readers of the National Computer Conference (NCC'87), which will be held in McCormick Place, Chicago, IL on June 15 – 18, 1987. The National Computer Conference provides a forum where significant developments and trends in technology are announced, discussed and displayed. NCC program includes both exhibits and presentations. For more information contact:

NCC '87
AFPIS
1899 Preston White Dr.
Reston, VA 22091
1(800)NCC–1987

### 4040 Drive Internals

Depending on reader response, a book could soon become available that uncovers, for the very first time, all inner details of the Commodore 4040 drive. Within this vast tome of knowledge will be found an in depth and documented look into the Floppy Disk Controller RAM and ROM, the Interface Processor RAM and ROM, plus theory on how it all fits together. A useful book for specific occasions. The book is close to completion right now, but reader response is required to determine if full production would be worth while. If you are at all interested, and would like to be kept informed of the book's progress, then send a note today to the following address. If the 4040 book is successful, then an 8050, 8250, 9060 and 9090 will follow.

Hillaire Gagne
1074 Webbwood Drive
Sudbury, Ontario, Canada
P3C–3B7

### GEOS Programming Guide

The OpCode Factory will be offering a technical programming guide to GEOS during the Spring of 1987. If you are interested in having more information on GEOS please send your name, address, and phone number to David Martin in care of:

The OpCode Factory
1417 South Heron Drive
Seabrook, Texas, 77586

### New Commodore Business Magazine

Money Machine is a new Commodore magazine introduced at the last LA Commodore Show. It is targeted at Commodore 64, 128 and Amiga owners who are using their computers for home applications and/or for small or home business management.

Money Machine is a bi-monthly magazine that retails for $3.95 US per copy in the stores, and is distributed by:

Redwood Distributing
P.O. Box 6609
San Mateo, CA 94403
(415)579-5506

Subscriptions are available from the publisher at the rate of $16.00 US for six issues. Contact:

Paula Vineyard
P.O. Box 2618
2142 E. Silver Springs Blvd.
Ocala, FL 32678
(904)622-1022

**Genealogy Software from ByteWare**

ByteWare has been known for developing genealogy software for Commodore computers. Their product, designed to ease the genealogist's record keeping tasks, was originally created for the C-64. Later, versions for Plus/4 and C-128 computers were developed. Most recently, the manufacturer has announced the release of the Genealogy package for the PET computers. Programs in both 4040 and 8050 formats are available.

For more information, sample sheets and prices for various programs, interested genealogists should contact:

ByteWare
Maple City Software
906 West 6th Avenue
Monmouth, IL 61462

**The MailRoom v2.1 for the Commodore 64**

Version 2.1 manages as many one-disk mailrooms as you may need, each with up to five distinct mailing lists containing up to 2000 labels indexed by name, city and ZIP code for 30-second retrieval of all matching labels. The user can define up to five non-exclusive sub-lists per disk, with up to three mutually-exclusive categories, for use in printing labels (in ZIP code order) or indices (by name, city or ZIP); select labels from any combination of lists with wild-card (*) search parameters; and enter listings first-name-first or last-name-first for first-name-first labels.

The program automatically checks new entries for matching names or addresses, allowing instant editing of an existing matching label with automatic update of indices and supports wild-card functions for edit and delete oprations or simple searches. With a second drive, The MailRoom will automatically merge two mailing lists or allow selective merges with editing.

The MailRoom for the C64, $49.95 plus $3.00 shipping and handling; or demo disk, $5.00. If Version 2.1 doesn't meet your needs, The DiskWorx will customize field length, disk capacity, and other features to your specifications for an additional $20.00. Specify address to be used for test label when ordering, make checks payable to David Shiloh at

The DiskWorx
1011 Vallley River Way, Suite 155
Eugene, Oregon 97401

**Spence XP BBS for C64 - $10**

The Spence XP BBS is a public domain BBS program for the C64. The program supports the 1650, Mitey Mo and 1670 modems. Features of the BBS are: 300/1200 baud, 0-10 directories, message base with 1-8 categories, bulletins, Punter and Xmodem protocol, runs in basic with ML routines, selectable drive configuration as well as online sysop commands.

Although the program is public domain we can mail you a copy with our 18 page manual. Send cheque or money order for $10 (Ontario residents add 7% PST) payable to ConText Publishing:

Spence XP BBS
c/o ConText Publishing
3092 Danforth Ave. Suite D
Scarborough Ont.
M1L 1B1

**SpeedScript Updated for the C-128**

SpeedPlus-128 from Lindon Enterprises converts a C-64 copy of SpeedScript 3.X into a full-featured 80-column C-128 version with 64K text memory and 20K erase buffer, all for use in 128 operating mode. The program also adds new features to SpeedScript, including justification, 12-value assignable tab, 2-column/2-side printing, word wrap toggle, selectable print-out, over 26 print commands programmable with up to 16 values each, window screen preview of documents for all margins and page lengths, secondary address change to "0" or "7" while printing, insertion of text files within a document, screen display of up to 26 help files from a single disk without loss of text in memory, changeable command line and screen display character color, and adjustable screen display of text for increased typing speed. SpeedPlus-128 comes with two help files: a 1986-87 calendar and a reference list of SpeedScript/SpeedPlus commands.

SpeedPlus-128 is available by mail order for $29.95 (US), including shipping and handling charges, from:

Lindon Enterprises
P.O. Box 773
Elm Grove
WI 53122

**Disk-2-Disk from CCS**

Central Coast Software announces Disk-2-Disk, which transfers C-64/C-128 files to and from AmigaDOS. (This disk-to-disk file transfer utility transfers SEQ, REL and USR files to the Amiga.)

Disk-2-Disk supports the 1541/4040 and 1570/1571 disk formats including 1541 "flippies". The program converts Commodore PET ASCII to AmigaDOS standard ASCII and vice versa. Disk-2-Disk formats 1541 and 1571 diskettes, runs under either the Intuition or CLI interfaces, supports AmigaDOS style wild cards in file names, provides TYPE and DELETE (Scratch) commands and permits renaming of files where file name restrictions occur.

Disk-2-Disk includes VALIDATE BAM and CHCK Disk utilities as well as a BASDIF utility to find and flag dialect differences in BASIC files.

Disk-2-Disk requires a standard Amiga with an Amiga model 1020 external 5.25" disk drive. $49.95 US through CCS and Amiga dealers. For further information, contact:

Central Coast Software
268 Bowie Drive
Los Osos, CA 93402
(805)528-4906

**New Interface for C-64/C-128 and IBM PC**

TecTrans-W.Guertzgen is the distributor for a new interface that allows connection of C-64/C-128 computers to IBM PC computers, or to any other computer, modem or printer with an RS232C interface. The interface, which is called 98084, is connected to the serial port of the C-64/C-128 and to the RS232C port of the IBM computers. The data can be transferred from the C-64/128 computer to the IBM computer or vice versa.

The interface comes with a 64 KByte buffer divided into a 32 KByte input buffer and a 32 KByte output buffer. There are no DIP switches on board - all adjustments for the interface are done by commands to an EEPROM into a non-volatile memory (the memory is programmable as often as the user likes).

**Technical data:** Baud rate: 225 to 57600. Databits: 7 or 8. Stopbits: 1 or 2. Parity none, even, odd, Handshake hardware or XON/XOFF. Connectors: IBM RS232 (DB25 female), Commodore serial connector. Power supply: from the cassette port of the C-64/128 or a separate power supply ($15.00 US - optional).

The interface package contains all cables to the computers and instructions for the user. The price for the product is $149.00 US (CA residents add tax 6.5%, shipping charges in U.S. are $4.00 on all orders). More information is available from:

TecTrans-W.Guertzgen
6925 Rosemead Blvd,
San Gabriel, CA 91775
(818)285-3121

**The Microtroll**

The Microtroll is the newest product from Slide Mountain Systems. It has been described by the manufacturer as a "customizable, real world interface" designed to work with a C-64 or C-128 (in C-64 mode).

The Microtroll is both hardware and software expandable. The Microtroll can be used to create home monitor systems, greenhouse control systems, robotics systems, automated test equipment, laboratory test and data gathering systems, manufacturing control systems or as an aid in teaching electronics.

The hardware consists of the main circuit board, a small interface board that plugs into the computer's cartridge port, and an AC power supply.

The software consists of a nearly 8K Operating System of Basic and Machine language routines, which auto-boots on power up. This program displays all input and output info, allows control of input from the keyboard, and is designed to link to user generated programs as well. Additional software is available on disk - MATT'S Microtroll Tools, disk 1, which can be purchased for $5.00 US from the manufacturer.

The Microtroll package includes extensive documentation. It can be ordered direct from the factory for $180.00 US plus $5.00 shipping and insurance. Microtroll carries a 90-day factory warranty on parts and labour for manufacturing defects. Payments should be made by cheque, money order, or COD. S.M.S. notes that personal cheques will slightly delay delivery. The manufacturer can be reached by calling or writing to:

Slide Mountain Systems
P.O. Box 6481
Colorado Springs, CO 80934
(303)449-4783

**Command Center for Commodore computers**

Ketek has announced a Command Center, a space-saving cabinet specially designed for the C-64/128 and 64C computers. The Command Center consolidates a computer, two disk drives and a monitor into a compact enclosure, keeping all cables hidden, out of sight and reach. Some features of the cabinet include a main power control switch eliminating turning on/off all peripherals separately, a cooling fan to prevent overheating, and a built-in A.C. power strip with surge protection and line noise filtering. Other options available include a cartridge port extension and a modular telephone plug with its own on-line/off-line switch. To order the Command Center, contact:

Ketek
P.O. Box 203
Oakdale, Iowa 52319

For fast service, call 1-800-626-4582 toll free. In Iowa call 319 338-7123.


DISK 2 DISK™

# Converts C64/C128 Files to the Amiga!

**DISK-2-DISK**™ from Central Coast Software makes it easy and convenient to transfer C64/C128 files to and from the Amiga. DISK-2-DISK programs the Amiga model 1020 external 5.25" disk drive to read and write 1541/4040 and 1570/1571 disk formats including 1541 "flippies". You can even format a 1541 or 1571 diskette on your Amiga!

**DISK-2-DISK** converts Commodore/PET ASCII to AmigaDOS standard ASCII and vice versa. Use DISK-2-DISK to transfer word processing text files (such as PaperClip, SpeedScript and Pocket Writer) to and from the Amiga for use with popular Amiga word processors.

**DISK-2-DISK** includes a utility to find and flag dialect differences between Commodore Basic and Amiga Basic files.

**DISK-2-DISK** includes VALIDATE BAM and CHECK DISK utilities. VALIDATE BAM verifies the directory structure of the 1541/1571 diskette. CHECK DISK reads every block of a 1541/1571 diskette to detect diskette errors.

**DISK-2-DISK** sells for $49.95 plus $3 shipping and handling. CA residents add 6% sales tax. Telephone orders welcome. Dealer inquires invited.

CCS

VISA

**Central Coast Software**™

268 Bowie Drive, Los Osos, CA 93402 805 / 528-4906

MasterCard

Trademarks: Amiga, AmigaDOS, Commodore-Amiga, Inc.; PaperClip, Batteries Included; Pocket Writer, Digital Solutions, Inc.; DISK-2-DISK, Central Coast Software.

## Lincoln College
## Commodore Computer Camp

with

## JIM BUTTERFIELD
and other experts

**July 19-25, 1987**

Topics include:

- Amiga
- C-128
- Robotics
- Telecomputing
- Additional selected topics

For further information, contact:
Office of Continuing Education
Lincoln College
300 Keokuk
Lincoln, IL 62656
217/732-3155



# EXPAND YOUR COMMODORE TO PERFORM LIKE NEVER BEFORE

**Just plug in the Final Cartridge.**

**only $54.95** New Improved

**Does NOT use existing memory**
The first completely external operating system created specifically for the Commodore 64, 64C, and 128 (in C64 mode).

**Upgrades hardware and software**
Takes the place of at least 6 separate devices, **1)** Disk Turbo–*6 times faster* loading and saving; **2)** Preprogrammed Function Keys–eliminate long, tedious command sequences for many commands, **3)** Extended Machine Language Monitor–with relocated load-scrolling up and down, bankswitching, and more; **4)** Printer Interface–prints all Commodore graphics plus screen-dump utility; **5)** Basic Tool Kit–*Auto Line Numbering, Delete* large program blocks with one touch, *Old* recovers accidentally-deleted programs, *Renumbering, Find, Help* debugs system, *Disk Append* adds new programs to existing files; **6)** Make backup copies of any software program.

**Other Convenient features**
**Freezer**– •16 sub menus • color changes • 4 resets • centronics/serial screendumps • print vector setting • reverse printing • stops and continues almost every program • allows total backup to disk or tape automatically • creates one file on disk or tape • freezes 4 to 6 times faster than dedicated freezers • game killer
**Screendump Capability**–Prints low-res, high-res and multicolor • prints full page • prints from games and more
**Keyboard Extras**–Delete parts of lines • move cursor • operates your printer as a typewriter

**BONUS! Additional 24K extra RAM for basic programs**

**10 Day Money-back Guarantee, Full year warranty**

**SPECIAL!**
Commodore to Centronics printer cable–$19.95

**Payments to:**
**H & P Computers**
- Bank or Certified Check, Personal Check, Money Order, Visa or MasterCard and C.O.D.
- Add $3.00 for shipping and handling
- NJ and NY residents add appropriate sales tax
- Dealer, distributor, user group inquiries welcome

H&P Home Personal COMPUTERS OF AMERICA

# THE FINAL CARTRIDGE®

154 Valley Street, South Orange, New Jersey 07079 (201) 763-3946



## EXPAND YOUR *AMIGA*™
## INTERNALLY
(No Soldering)
## BY ONE MEGABYTE !
(1,000K or 1,000,000 Bytes)
With the
## "INSIDER!!!"

- The **BEST** Internal RAM expansion.
- One Megabyte of FAST Auto-Config RAM.
- No Wait States.
- Battery Backed-Up Clock/Calendar Chip.
- Compatible with Sidecar™.
- Compatible with Most other External RAM.
- One Year Warranty.

Suggested Retail **$595.00** Canadian Funds

Canadian Distributor
**DESIGNTECH BUSINESS SYSTEMS INC.**
#304-850 Burrard Street
Vancouver, BC V6Z 2J1
(604) 669-1855

Certified Cheque or MO.
Dealer Enquiries Invited

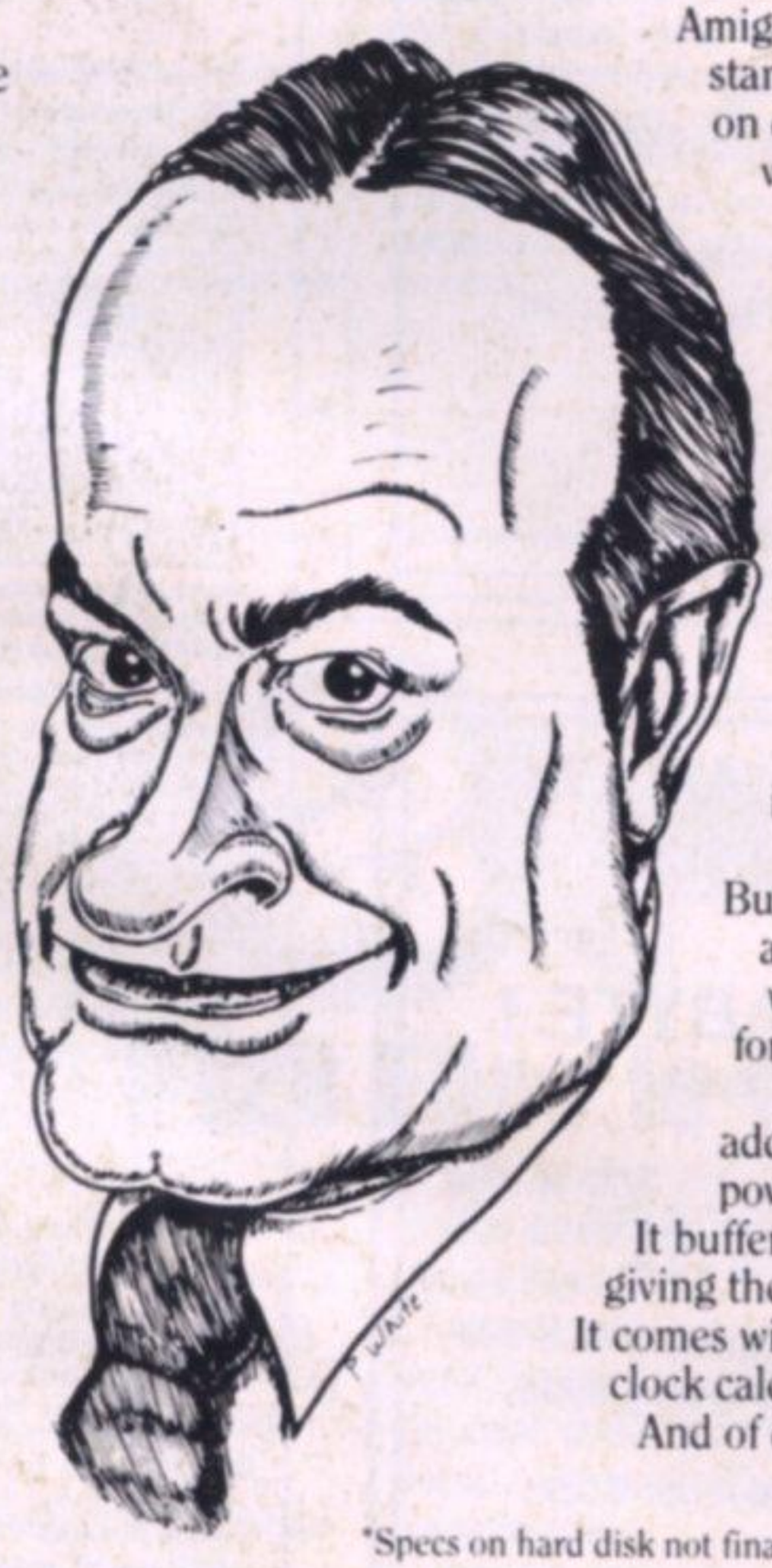
# *Thanks for the* **Memories**

## 2 MEG RAM EXPANSION

Comspec's second generation AX2000 RAM BOARD.
Software developers worldwide have been using it for over a year. Now so can *you*.
The AX2000 provides "fast" RAM, giving you more room for program and data storage, faster program execution and fewer time-consuming disk accesses.
The AX2000 provides a full 2 Megabytes of memory *now*.
The AX2000 is fully compatible with *all* standard Amiga products. Some boards aren't.
The AX2000 is auto configuring*. All you do is plug it into your Amiga and turn it on. Nothing else is needed.
Full "pass through" allows for complete peripheral expansion.
It'll even survive a "soft" reset.

*Using Workbench 1.2

Amiga is a registered trademark of Commodore Business Machines.



## HARD DISK

Finally, a hard disk drive for the Amiga made to the same exacting standards as the AX2000. It runs on every voltage from 95 to 260 volts. It uses 50 or 60 Hz. It's available simply as a SCSI host adaptor, a SCSI host adaptor and controller, or as a complete 20 Meg. hard drive. In fact the machine accepts hard drives starting from 10 Meg and up. The hard drive can be combined with the AX2000 to provide up to 8 Meg of RAM memory. Adaptability is the definition of Comspec's new hard disk drive.
But the hard disk is more than adaptable. It comes standard with features you won't soon forget. A built-in power supply gives the Amiga an additional 2 amps of necessary power for use with peripherals.
It buffers the Amiga expansion bus, giving the computer better flexibility.
It comes with a built-in SCSI port. The clock calendar has a battery back-up.
And of course it's auto-configuring.

*Specs on hard disk not finalized.

**COMSPEC**

153 Bridgeland Ave., Unit #5
Toronto, Ontario Canada M6A 2Y6
(416) 787 0617

Distributed worldwide by

**Comspec**
Toronto, Canada
(416) 787 0617

**Air-Stat of Canada Inc.**
Markham, Canada
(416) 477 9440 1 800 268 3314

**Run Informatique**
Paris, France
33-1-45 81 5144

**Microtron**
Pieterlen, Switzerland
41-32 87 2429

**Ingeniorfirmaet Finn Jacobson**
Bagsvaerd, Denmark
45-2-44 0488

**Precision Software Limited**
Surrey, England
01-330 7166

**Nerika Australia Proprietary Ltd.**
Sidney, Australia
957-4778

**Southern Technologies Inc.**
Dallas, Texas
United States
(214) 247-7373